# Applying Abstract Acceleration to (Co-)Reachability Analysis of Reactive Programs

Peter Schrammel Bertrand Jeannet

*INRIA Grenoble - Rhône-Alpes*
*655 Avenue de l'Europe*
*38330 Montbonnot St-Martin*
*France*

**Abstract**

Acceleration methods are commonly used for computing precisely the effects of loops in the reachability analysis of counter machine models. Applying these methods on synchronous data-flow programs, *e.g.* Lustre programs, requires to deal with the non-deterministic transformations due to numerical input variables. In this article we address this problem by extending the concept of abstract acceleration of Gonnord et al. to numerical input variables. Moreover, we describe the dual analysis for co-reachability. We compare our method with some alternative techniques based on abstract interpretation pointing out its advantages and limitations. At last, we give some experimental results.

*Key words:* Static Analysis, Acceleration, Abstract Interpretation, Linear Relation Analysis.

## 1. Introduction

This article considers the reachability analysis of non-recursive, numerical programs represented by symbolic automata manipulating numerical variables, as illustrated in Fig. 1(a). More specifically, we focus on techniques enabling a precise analysis of self-loops that can be smoothly combined with methods for general numerical programs. For instance, considering the program of Fig. 1(a) with the set $X_0$ of initial values for the

⋆ This work was supported by the INRIA large-scale initiative Synchronics and the ANR project Asopt.

*Email addresses:* peter.schrammel@inria.fr (Peter Schrammel ), bertrand.jeannet@inria.fr (Bertrand Jeannet).

*URLs:* http://pop-art.inrialpes.fr/people/schramme (Peter Schrammel ), http://pop-art.inrialpes.fr/people/bjeannet (Bertrand Jeannet).

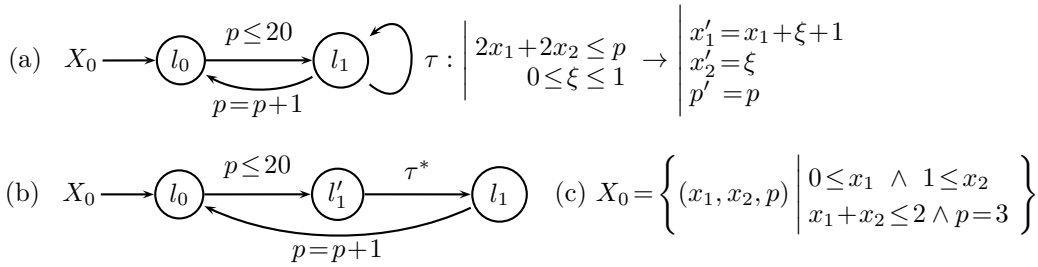


Fig. 1. Example program (a), transformed program (b) where $\tau^*$ denotes the transitive closure of the transition $\tau$, and set of initial states (c).

state variables $x_1, x_2, p$, we want to compute the possible values of these state variables at locations $l_0$ and $l_1$. This article proposes *abstract acceleration methods* (as introduced by Gonnord and Halbwachs (2006)) to capture precisely the effect of the inner loop labelled by $\tau$ on a set of states $X \in \mathbb{R}^n$. It shows how these techniques integrate nicely with less precise but more general methods applying to programs with control structure involving nested loops and unstructured cycles.

Our ultimate motivation is the reachability analysis of data-flow synchronous programs manipulating Boolean and numerical variables, which are for instance specified with the Lustre language (Caspi et al. (1987)). Applications of such reachability analyses are for instance the verification of safety properties (Halbwachs et al. (1993)) or model-based testing (Jeannet et al. (2005)).

We first give an overview of existing methods for the reachability analysis of the systems we consider, before detailing the original contributions of this article.

*Abstract interpretation and acceleration.* Since the reachability problem is not decidable for numerical programs that encode two-counter automata (Minsky (1961)), two main approaches have been studied to overcome this fundamental limitation:

(1) Abstract interpretation techniques (Cousot and Cousot (1977, 1992a)) always terminate with a sound over-approximation of the reachability set.
(2) Acceleration techniques (*e.g.* Leroux (2003); Bardin et al. (2003, 2005)) compute the exact reachability set for a restricted class of programs, *e.g.*, for programs with certain affine tests and assignments. However, there is no guarantee for termination.

In both approaches, the set of reachable states is obtained by solving iteratively an equation of the form $X = X_0 \cup \mathit{post}(X)$ where $X$ is a set of states, $X_0$ the initial set, and *post* the postcondition operator associated with the program.

*Abstract interpretation* is a classical method for analyzing programs with infinite state spaces. The key idea is to approximate sets of states $X$ by an element $Y$ of an *abstract domain*. Two classical abstract domains for *numerical invariants* $X \in \wp(\mathbb{R}^n)$ are the domain of *convex polyhedra* $\mathit{Pol}(\mathbb{R}^n)$ (Cousot and Halbwachs (1978)), that are conjunctions of linear inequalities $\bigwedge_i (\boldsymbol{a_i x} \leq b_i)$ and the *linear congruences* domain (Granger (1991); Bagnara et al. (2006)) that represents conjunctions of linear congruences $\bigwedge_i (\boldsymbol{a_i x} = b_i \bmod c_i)$. An approximation of the reachable set is computed by solving iteratively the equation $Y = Y_0 \sqcup \mathit{post}(Y)$ in the abstract domain. In order to ensure termination when the abstract domain contains infinitely increasing chains, an extrapolation operator called *widening* is applied, which induces additional over-approximations.

The idea of *acceleration* is to accelerate cycles labelled by a function $\tau$ in the control structure of a program by computing the effect of its reflexive and transitive closure $\tau^* = \bigcup_{k\geq 0} \tau^k$ on a set of states $X$ (the term "closure" refers to $\tau$ viewed as a relation). Applied to the program of Fig. 1(a), we obtain the program of Fig. 1(b). If the program is flat (*i.e.* it does not contain nested loops) and all loops can be accelerated, then the method is complete. If the program contains nested loops as in Fig. 1(a), the method is not complete any more; the standard heuristics is to enumerate and accelerate cycles (which form an infinite set) in the hope of terminating with the smallest fixed point after a finite number of steps. The same remark applies if transition functions in some cycles are too expressive to be accelerated. Acceleration has been mostly applied to automata manipulating integer variables using Presburger arithmetic (Fribourg and Olsén (1997); Finkel and Leroux (2002); Bardin et al. (2003)), or FIFO queues using subclasses of regular expressions (Boigelot and Godefroid (1997); Abdulla et al. (2004)).

Widening basically extrapolates the limit of a sequence of abstract invariants without referring to the program that generates them, whereas acceleration uses the structure of the program to perform an exact extrapolation. Gonnord and Halbwachs (2006) have proposed the concept of *abstract acceleration* which combines these approaches: wherever possible, simple loops are accelerated *in the abstract domain*, and in any other cases (multiple self-loops, nested loops, too expressive transitions) one resorts to the use of widening to guarantee the convergence of the fixed point computation at the cost of over-approximations.

*Applying acceleration to reactive programs.* Many acceleration techniques such as those introduced by Leroux (2003); Bardin et al. (2003); Gonnord and Halbwachs (2006) consider automata with transition functions in the form of guarded actions

$$\tau : \underbrace{g(\boldsymbol{x})}_{guard} \rightarrow \underbrace{\boldsymbol{x'} = \boldsymbol{f}(\boldsymbol{x})}_{action} \qquad \boldsymbol{x}, \boldsymbol{x'} \in \mathbb{R}^n \tag{1}$$

However, reactive programs such as LUSTRE data-flow programs interact with their environment: at each computation step they have to take into account the value of *input* variables, which typically correspond to values acquired by sensors.

Boolean input variables can be encoded in an automaton by finite non-deterministic choices but numerical input variables demand a more specific treatment. Indeed, they induce transition functions of the form

$$\tau : g(\boldsymbol{x}, \boldsymbol{\xi}) \rightarrow \boldsymbol{x'} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\xi}) \qquad \boldsymbol{x}, \boldsymbol{x'} \in \mathbb{R}^n \quad \boldsymbol{\xi} \in \mathbb{R}^p \tag{2}$$

that depends on both state variables $\boldsymbol{x}$ and numerical input variables $\boldsymbol{\xi}$. This article addresses specifically this point, by extending the abstract acceleration concept introduced by Gonnord and Halbwachs (2006) to systems with numerical inputs. The methods developped in this article can also be seen as an alternative to acceleration methods-based on the computation of transitive closures of affine *relations* (rather than functions) (Beletska et al. (2009); Bozga et al. (2010)). Indeed, transition functions with inputs can be viewed as relations between states defined as

$$R(\boldsymbol{x}, \boldsymbol{x'}) = \big(\exists \boldsymbol{\xi} : g(\boldsymbol{x}, \boldsymbol{\xi}) \wedge \boldsymbol{x'} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\xi})\big) \tag{3}$$

*Contributions and outline.* Our first two contributions are extensions of the abstract acceleration concept as introduced by Gonnord and Halbwachs (2006) in two directions:

(i) We consider open systems, *i.e.*, with numerical inputs, instead of closed systems; in particular we show how to accelerate loops where translations and resets depend on *inputs*, provided that the guard of the loop constrains separately state and input variables.

(ii) We also extend abstract acceleration techniques from forward (reachability) analysis to backward (co-reachability) analysis. In consequence, it possible to apply the abstract acceleration concept to related, but slightly different problems in verification, such as parameter synthesis for example. Moreover, experience shows that combining forward and backward analyses is very useful in the context of abstract interpretation (*e.g.* Jeannet (2003)).

A third contribution is (iii) a detailed comparison of the abstract acceleration approach to the derivative closure approach of Ancourt et al. (2010), which is related to methods based on transitive closures of relations. This article extends the results presented by Schrammel and Jeannet (2010) with the contributions (ii) and (iii).

After some preliminaries in Section 2 about the considered program model, operations on convex polyhedra and the general analysis framework that we use for verification, we recall the main results of abstract acceleration in Section 3. Section 4 extends abstract acceleration techniques to systems with numerical input variables, in the context of forward analysis. Section 5 extends these results to backward analysis. Section 6 is dedicated to a comparison with other methods. Section 7 gives an overview and some experimental results on how to apply abstract accleration to logico-numerical programs. After a discussion of further related work in Section 8 we conclude in Section 9.

## 2. Analysis of Logico-Numerical Programs

*Program model.* We assume a set $V$ of variable names. We consider in this article programs modeled as numerical automata $(L, \boldsymbol{x}, \boldsymbol{\xi}, l_0, X_0, \mathcal{T})$ where

- $L$ is a finite set of locations, $\boldsymbol{x} \in V^n$ a vector of real-valued *state* variables, and $\boldsymbol{\xi} \in V^p$ a vector of real-valued *input* variables;
- $l_0$ is the initial location and $X_0 \subseteq \mathbb{R}^n$ is the set of initial values for state variables $\boldsymbol{x}$ at location $l_0$;
- $\mathcal{T}$ is a finite set of transitions of the form $t : (l, l', \tau)$ where $l$ and $l'$ are respectively the origin and destination locations, and $\tau$ is a (partial) transition function of the form $g(\boldsymbol{x}, \boldsymbol{\xi}) \to \boldsymbol{x'} = f(\boldsymbol{x}, \boldsymbol{\xi})$.

An execution of such a system is a sequence $(l_0, \boldsymbol{x}_0) \xrightarrow{t_0, \boldsymbol{\xi}_0} \dots (l_k, \boldsymbol{x}_k) \xrightarrow{t_k, \boldsymbol{\xi}_k} \dots$ such that $\boldsymbol{x}_0 \in X_0$ and for any $k \geq 0$, $t_k = (l_k, l_{k+1}, \tau_k)$ and $\boldsymbol{x}_{k+1} = \tau_k(\boldsymbol{x}_k, \boldsymbol{\xi}_k)$.

This program model includes various models of counter automata (Comon and Jurski (1998); Bardin et al. (2005)). It can also be obtained from Lustre synchronous data-flow programs by (1) taking the output of their front-end compilation process, (2) performing on this output a partial evaluation (Jones et al. (1993)) of all Boolean state variables (which are then encoded in control locations), and (3) eliminating Boolean input variables using non-deterministic choices. The partition refinement mechanics implemented in the NBac tool (Jeannet (2003)) are capable of achieving this task and have been employed for connecting the Aspic tool (Gonnord (2007, 2009)) to Lustre, for example.

*Convex Polyhedra.* We use in this article the abstract domain of convex polyhedra for representing invariants on numerical variables. Convex polyhedra can be represented either as a conjunction of constraints denoted by

$$\mathbf{A}\boldsymbol{x} \leq \boldsymbol{b} \text{ with } \boldsymbol{x} \in \mathbb{R}^n, \mathbf{A} \in \mathbb{R}^{m\times n}, \boldsymbol{b} \in \mathbb{R}^m$$

or as the set of vectors generated by a vector of vertices $\mathbf{V} = (\boldsymbol{v}_1 \dots \boldsymbol{v}_p) \in \mathbb{R}^{n\times p}$ and rays $\mathbf{R} = (\boldsymbol{r}_1 \dots \boldsymbol{r}_q) \in \mathbb{R}^{n\times q}$, denoted by $(V, R)$:

$$\{\boldsymbol{x} \mid \exists \boldsymbol{\lambda}, \boldsymbol{\mu} \geq 0 \ : \ \textstyle\sum_i \lambda_i = 1 \wedge \boldsymbol{x} = \mathbf{V}\boldsymbol{\lambda} + \mathbf{R}\boldsymbol{\mu}\}$$

We can convert from one representation to the other one (see (Fukuda and Prodon (1996)). We will use the same notation for polyhedra $X$ interexchangeably for both the predicate $X(\boldsymbol{x}) = (\mathbf{A}\boldsymbol{x} \leq \boldsymbol{b})$ and the set $X = \{\boldsymbol{x} \mid \mathbf{A}\boldsymbol{x} \leq \boldsymbol{b}\}$. We briefly remind some classical operations described *e.g.* by Halbwachs et al. (1997):

The constraint representation is used for computing the intersection $\sqcap$ of two polyhedra (*i.e.* the conjunction of their constraints) and the inverse image $\tau^{-1}(X)$ of a polyhedron $X = (\mathbf{A}\boldsymbol{x} \leq \boldsymbol{b})$ by an affine assignment $\tau : \boldsymbol{x'} = \mathbf{C}\boldsymbol{x} + \boldsymbol{d}$, *i.e.* $\tau^{-1}(X) = (\mathbf{AC}\boldsymbol{x} \leq \boldsymbol{b} - \mathbf{A}\boldsymbol{d})$.

The generator representation is used for computing the convex hull (union) $\sqcup$ of two polyhedra (*i.e.* the union of their generators), the projection of dimensions, and the image $\tau(X)$ of a polyhedron $X = (V, R)$ by an affine assignment $\tau : \boldsymbol{x'} = \mathbf{C}\boldsymbol{x} + \boldsymbol{d}$: $\tau(X) = \big((\mathbf{CV} + (\underbrace{\boldsymbol{d} \dots \boldsymbol{d}}_{p \text{ times}})), \mathbf{CR}\big)$.

Mind that the intersection of two convex polyhedra is again a convex polyhedron: $X \cap Y = X \sqcap Y$. In contrast, the union of two convex polyhedra is not convex in general. For that reason the convex hull, *i.e.* an over-approximation, is used instead of the ordinary union for sets: $X \cup Y \subseteq X \sqcup Y$

The *time elapse* operation, defined as $X \nearrow D = \{\mathbf{x} + t\mathbf{d} \,|\, \mathbf{x} \in X, \mathbf{d} \in D, t \in \mathbb{R}^{\geq 0}\}$ can be implemented using the systems of generators $(V_X, R_X)$ and $(V_D, R_D)$ of the polyhedra $X$ and $D$: $(V_X, R_X \cup V_D \cup R_D)$ is a system of generators for $X \nearrow D$.

The *Minkowski sum* (see de Berg et al. (2008)) of two polyhedra $X = X_1 + X_2$ is defined by $X(\boldsymbol{x}) = \exists \boldsymbol{x}_1, \boldsymbol{x}_2 \ : \ (\boldsymbol{x} = \boldsymbol{x}_1 + \boldsymbol{x}_2) \wedge X_1(\boldsymbol{x}_1) \wedge X_2(\boldsymbol{x}_2)$.

$\top$ and $\bot$ denote the polyhedra $\mathbb{R}^n$ and $\emptyset$ respectively.

Concerning the complexity, the translation between the two representations may be exponential in the number of dimensions $n$ (see Fukuda and Prodon (1996)), hence any composition of basic operations requiring both representations has a worst-case exponential complexity.

Note that the operations defined on generators can also be computed using only systems of constraints and projection as shown by Benoy et al. (2005). However, this does not allow to improve the worst-case complexity.

## 3. Overview of Acceleration and Abstract Acceleration

As mentioned in the introduction, the idea of *acceleration* (Fig. 1) is to replace a self-loop $\tau$ by an ordinary transition $\tau^*$ that is the reflexive and transitive closure of $\tau$. *Abstract acceleration* introduced by Gonnord and Halbwachs (2006) and Gonnord (2007) relaxes exact acceleration in the sense that it aims at approximating the exact set $\tau^*(X)$ by a convex polyhedron $\tau^{\otimes}(X) \supseteq \tau^*(X)$ that is close to the convex hull of the exact

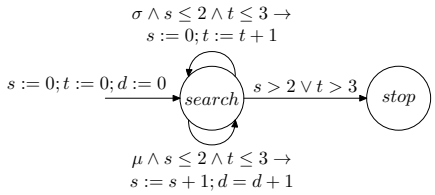


Fig. 2. Example: A robot car safety controller. The lower loop means: When a "meter" signal $\mu$ is received the speed estimation $s$ and the distance $d$ are incremented: this transition is a translation. On the contrary, the upper loop is a translation with resets: when a "second" signal $\sigma$ is received the speed estimation is reset and the time $t$ is incremented. If the speed is above $2m/s$ or no input signal has been received for $3s$ then the car is stopped.

set. This method is also inspired by the time elapse operator used in timed or in hybrid automata (Halbwachs et al. (1997)).

Following the notations of Section 2, a self-loop $\tau$ has the structure: $G \rightarrow A$ meaning "while guard $G$ do action $A$". Generally, acceleration methods for numerical variables $\boldsymbol{x}$ deal with transitions of the form

$$\mathbf{A}\boldsymbol{x} \leq \boldsymbol{b} \rightarrow \boldsymbol{x}' = \mathbf{C}\boldsymbol{x} + \boldsymbol{d} \tag{4}$$

where $\mathbf{A}\boldsymbol{x} \leq \boldsymbol{b}$ represents a conjunction of linear constraints defining a convex polyhedron, and $\boldsymbol{x}' = \mathbf{C}\boldsymbol{x} + \boldsymbol{d}$ is an affine transformation; $\mathbf{C}$ is a square matrix. A transition is called

- a *reset* iff $\mathbf{C}$ is the zero matrix,
- a *translation* iff $\mathbf{C}$ is the identity matrix,
- a *translation with resets* (or translation/reset) iff $\mathbf{C}$ is a diagonal matrix with zeros and ones only,
- a *periodic affine transformation* iff $\exists p > 0, \exists l > 0 : \mathbf{C}^{p+l} = \mathbf{C}^p$,
- a *general affine transformation* otherwise.

Existing acceleration methods do not address general affine transformations. We will not consider in this paper the case of periodic affine transformations, but we discuss the generalization of our results to this case in the conclusion.

Fig. 2 shows an example of a simplistic safety controller for a robot car (cf. Halbwachs et al. (1997); Ancourt et al. (2010)). The car should follow a given track, but in case it has lost the track it should keep on searching the track for a while before being stopped. This example illustrates two of the above types of transitions.

In the context of abstract acceleration, Gonnord and Halbwachs (2006) show that translations (Fig. 3) and translations with resets (Fig. 4) can be accelerated as follows:

**Theorem 1.** *Let $\tau$ be a translation $G \rightarrow \boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{d}$, where the guard $G$ is a convex polyhedron. Then, for every convex polyhedron $X$, the convex polyhedron*

$$\tau^{\otimes}(X) = X \sqcup \Big( \big( (X \sqcap G) \nearrow \{\boldsymbol{d}\} \big) \sqcap (G + \{\boldsymbol{d}\}) \Big)$$

*is a convex over-approximation of $\tau^*(X)$.*

**Theorem 2.** *Let $\tau$ be a translation with resets $G \rightarrow \boldsymbol{x}' = \mathbf{C}\boldsymbol{x} + \boldsymbol{d}$, where the guard $G$ is a convex polyhedron. Then, for every convex polyhedron $X$, the convex polyhedron*

$$\tau^{\otimes}(X) = X \sqcup \tau(X) \sqcup \Big( \big( (\tau(X) \sqcap G) \nearrow \{\mathbf{C}\boldsymbol{d}\} \big) \sqcap (G + \{\mathbf{C}\boldsymbol{d}\}) \Big)$$

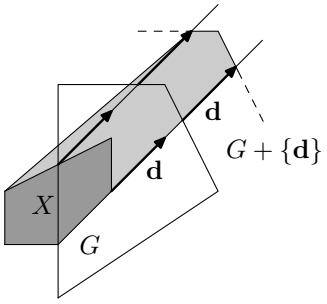


Fig. 3. Acceleration of a translation loop starting from $X$ (dark shadowed) resulting in $\tau^{\otimes}(X)$ (whole shadowed area).

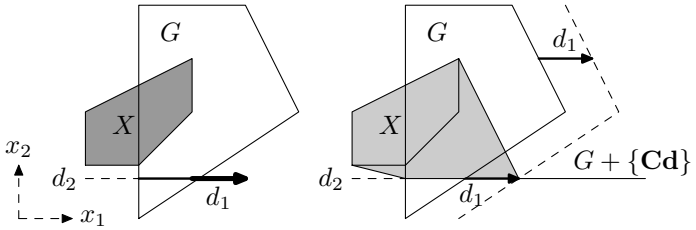


Fig. 4. Acceleration of a loop with translations/resets: On the left-hand side, the application of $\tau$ to $X$ – here, with $x_1' = x_1 + d_1$ and $x_2' = d_2$, yields a polyhedron (bold line including arrow) containing the reset values. The accelerated transition gives $\tau^{\otimes}(S)$ (shadowed) on the right-hand side.

*is a convex over-approximation of* $\tau^*(X)$.

**Remark 1.** Theorem 2 exploits the property that a translation with resets to *constants* iterated $N$ times is equivalent to the same translation with resets, followed by a pure translation iterated $N-1$ times. Hence the structure of the obtained formula

**Remark 2.** Ideally, $\tau^{\otimes}(X)$ as defined in Theorems 1 and 2 should be the best over-approximation of $\tau^*(X)$ by a convex polyhedron. This is not the case as shown by the following example in one dimension. Let $X = [1,1]$ and $\tau : x_1 \leq 4 \rightarrow x_1' = x_1 + 2$. $\tau^{\otimes}(X) = [1,6]$, whereas the best convex over-approximation of $\tau^*(X) = \{1,3,5\}$ is the interval $[1,5]$. This is because the operations involved in the definition of $\tau^{\otimes}(X)$ manipulate dense sets and do not take into account arithmetic congruences. In this article we will not improve in this respect, but we will point out in our proofs where this *dense* approximation takes place, and we discuss in Section 9 how the linear congruences abstract domain mentioned in the introduction can be exploited to improve on this point.

## 4. Abstract Acceleration with Numerical Inputs

We now extend numerical abstract acceleration by numerical input variables $\boldsymbol{\xi}$. This means that we consider transitions of the form

$$\underbrace{\begin{pmatrix}\mathbf{A} & \mathbf{L}\\ 0 & \mathbf{J}\end{pmatrix}\begin{pmatrix}\boldsymbol{x}\\ \boldsymbol{\xi}\end{pmatrix} \leq \begin{pmatrix}\boldsymbol{b}\\ \boldsymbol{k}\end{pmatrix}}_{\mathbf{A}\boldsymbol{x}+\mathbf{L}\boldsymbol{\xi}\leq\boldsymbol{b}\ \wedge\ \mathbf{J}\boldsymbol{\xi}\leq\boldsymbol{k}} \rightarrow \boldsymbol{x'} = \underbrace{\begin{pmatrix}\mathbf{C} & \mathbf{T}\end{pmatrix}\begin{pmatrix}\boldsymbol{x}\\ \boldsymbol{\xi}\end{pmatrix} + \boldsymbol{u}}_{\mathbf{C}\boldsymbol{x}+\mathbf{T}\boldsymbol{\xi}+\boldsymbol{u}} \tag{5}$$

Note that the 0 in the matrix of the guard does not imply a loss of generality.

A fundamental observation is that any general affine transformation without inputs $\mathbf{A}\boldsymbol{x} \leq \boldsymbol{b} \rightarrow \boldsymbol{x'} = \mathbf{C}\boldsymbol{x} + \boldsymbol{d}$ can be expressed

- as a "reset with inputs" $(\mathbf{A}\boldsymbol{x} \leq \boldsymbol{b} \wedge \boldsymbol{\xi} = \mathbf{C}\boldsymbol{x} + \boldsymbol{d}) \rightarrow \boldsymbol{x'} = \boldsymbol{\xi}$,
- as well as a "translation with inputs" $(\mathbf{A}\boldsymbol{x} \leq \boldsymbol{b} \wedge \boldsymbol{\xi} = (\mathbf{C}-\mathbf{I})\boldsymbol{x} + \boldsymbol{d}) \rightarrow \boldsymbol{x'} = \boldsymbol{x} + \boldsymbol{\xi}$, where $\mathbf{I}$ is the identity matrix

This means that there is no hope to get precise acceleration for such resets with inputs, unless we know how to accelerate precisely general affine transformations without inputs, which is out of the scope of the current state of the art.

Nevertheless, we can accelerate transitions with inputs when the constraints on the state variables do not depend on the inputs, *i.e.*, when $\mathbf{L} = 0$ in Eqn. (5) *i.e.*, the guard is of the form $\mathbf{A}\boldsymbol{x} \leq \boldsymbol{b} \wedge \mathbf{J}\boldsymbol{\xi} \leq \boldsymbol{k}$. We call the resulting guards *simple guards*. We provide in Section 4.3 a weaker over-approximation of the result for general guards.

### *4.1. Translations with inputs and simple guards*

Translations with inputs and simple guards are defined by

$$\underbrace{\begin{pmatrix}\mathbf{A} & 0\\ 0 & \mathbf{J}\end{pmatrix}\begin{pmatrix}\boldsymbol{x}\\ \boldsymbol{\xi}\end{pmatrix} \leq \begin{pmatrix}\boldsymbol{b}\\ \boldsymbol{k}\end{pmatrix}}_{\mathbf{A}\boldsymbol{x}\leq\boldsymbol{b}\,\wedge\,\mathbf{J}\boldsymbol{\xi}\leq\boldsymbol{k}} \rightarrow \boldsymbol{x'} = \underbrace{\begin{pmatrix}\mathbf{I} & \mathbf{T}\end{pmatrix}\begin{pmatrix}\boldsymbol{x}\\ \boldsymbol{\xi}\end{pmatrix} + \boldsymbol{u}}_{\boldsymbol{x}+\mathbf{T}\boldsymbol{\xi}+\boldsymbol{u}}$$

.

The first step we perform is to reduce such a translation with inputs to a *polyhedral translation* denoted $\tau : G \rightarrow \boldsymbol{x'} = \boldsymbol{x} + D$ and with the meaning $\tau(X) = (X \sqcap G) + D$.

**Proposition 1.** A translation $\tau$ with inputs and a simple guard is equivalent to a polyhedral translation defined by

$$G \rightarrow \boldsymbol{x'} = \boldsymbol{x} + D \quad \text{with} \quad D = \{\boldsymbol{d} \mid \exists \boldsymbol{\xi} : \boldsymbol{d} = \boldsymbol{T}\boldsymbol{\xi} + \boldsymbol{u} \wedge \boldsymbol{J}\boldsymbol{\xi} \leq \boldsymbol{k}\}$$

($D$ can be computed by standard polyhedra operations.)

**Proof.**

$\boldsymbol{x'} \in \tau(X)$
$\Leftrightarrow \exists \boldsymbol{x} \in X, \exists \boldsymbol{\xi} \;:\; \mathbf{A}\boldsymbol{x} \leq \boldsymbol{b} \wedge \mathbf{J}\boldsymbol{\xi} \leq \boldsymbol{k} \wedge \boldsymbol{x'} = \boldsymbol{x} + \mathbf{T}\boldsymbol{\xi} + \boldsymbol{u}$
$\Leftrightarrow \exists \boldsymbol{x} \in X \sqcap G, \exists \boldsymbol{\xi}, \exists \boldsymbol{d} \;:\; \mathbf{J}\boldsymbol{\xi} \leq \boldsymbol{k} \wedge \boldsymbol{d} = \mathbf{T}\boldsymbol{\xi} + \boldsymbol{u} \wedge \boldsymbol{x'} = \boldsymbol{x} + \boldsymbol{d}$
$\Leftrightarrow \exists \boldsymbol{x} \in X \sqcap G, \exists \boldsymbol{d} \in D \;:\; \boldsymbol{x'} = \boldsymbol{x} + \boldsymbol{d}$ with $D = \{\boldsymbol{d} \mid \exists \boldsymbol{\xi} \;:\; \mathbf{J}\boldsymbol{\xi} \leq \boldsymbol{k} \wedge \boldsymbol{d} = \mathbf{T}\boldsymbol{\xi} + \boldsymbol{u}\}$
□

We now generalize Theorem 1 from ordinary translations to polyhedral translations.

**Proposition 2.** Let $\tau$ be a polyhedral translation $G \rightarrow \boldsymbol{x'} = \boldsymbol{x} + D$. Then, the set

$$\tau^{\otimes}(X) = X \sqcup \tau\big((X \sqcap G) \nearrow D\big)$$

is a convex over-approximation of $\tau^*(X)$.

**Proof.** $\boldsymbol{x'} \in \bigcup_{k\geq 1} \tau^k(X) \Leftrightarrow \boldsymbol{x'} \in \tau(\bigcup_{k\geq 0} \tau^k(X))$

$$\Leftrightarrow \exists k \geq 0, \exists \boldsymbol{x}_0 \in X, \exists \boldsymbol{x}_k, \exists \boldsymbol{d}_1 \ldots \boldsymbol{d}_k \in D \;:\; \begin{cases} \boldsymbol{x'} \in \tau(\boldsymbol{x}_k) \\ \boldsymbol{x}_k = \boldsymbol{x}_0 + \sum_{j=1}^{k} \boldsymbol{d}_j \\ G(\boldsymbol{x}_0) \;\wedge\; \forall k' \in [1,k] \;:\; G(\boldsymbol{x}_0 + \sum_{j=1}^{k'} \boldsymbol{d}_j) \end{cases}$$

$\Leftrightarrow \exists k \geq 0, \exists \boldsymbol{x}_0 \in X, \exists \boldsymbol{x}_k, \exists \boldsymbol{d} \in D \;:\; \boldsymbol{x'} \in \tau(\boldsymbol{x}_k) \;\wedge\; \boldsymbol{x}_k = \boldsymbol{x_0} + k\boldsymbol{d} \;\wedge\; G(\boldsymbol{x}_0) \;\wedge\; G(\boldsymbol{x}_k)$
(because $D$ and $G$ are convex, see Remark 3)

$\Rightarrow \exists \alpha \geq 0, \exists \boldsymbol{x}_0 \in X, \exists \boldsymbol{x}_k, \exists \boldsymbol{d} \in D \ : \ \boldsymbol{x}' \in \tau(\boldsymbol{x}_k) \ \wedge \ \boldsymbol{x}_k = \boldsymbol{x}_0 + \alpha \boldsymbol{d} \ \wedge \ G(\boldsymbol{x}_0)$
(dense approximation; $G(\boldsymbol{x}_k)$ implied by $\boldsymbol{x}' \in \tau(\boldsymbol{x}_k)$)
$\Leftrightarrow \exists \boldsymbol{x}_0 \in X \sqcap G, \exists \boldsymbol{x}_k \ : \ \boldsymbol{x}' \in \tau(\boldsymbol{x}_k) \ \wedge \ \boldsymbol{x}_k \in (\{\boldsymbol{x}_0\} \nearrow D)$
$\Leftrightarrow \boldsymbol{x}' \in \tau((X \sqcap G) \nearrow D)$
$\square$

Mind that the only approximation takes place in the line ($\Rightarrow$) where the integer coefficient $k-1 \geq 0$ is replaced by a real coefficient $\alpha \geq 0$. This is the technical explanation of Remark 2.

**Remark 3.** The convexity argument employed in the previous proof is based on the following fact: $G(\boldsymbol{x}) \wedge G(\boldsymbol{x} + \sum_{j=1}^{k} \boldsymbol{d}_j) \Rightarrow \exists \boldsymbol{d} \in D : G(\boldsymbol{x}) \wedge G(\boldsymbol{x} + \alpha \boldsymbol{d})$ where $\sum_{j=1}^{k} \boldsymbol{d}_j = \alpha \boldsymbol{d}, \alpha \geq 0, \boldsymbol{d}_j \in D$. First, any intermediate point $\boldsymbol{x} + \alpha \boldsymbol{d}$ must be in $G$, because $G$ is convex. Second, such a $\boldsymbol{d}$ exists, for example, choose $\boldsymbol{d} = \frac{1}{k} \sum_{j=1}^{k} \boldsymbol{d}_j$ which is obviously in $D$, because $D$ is convex.

**Remark 4.** One might think that Theorem 1 can be applied directly by accelerating the transition for each $\boldsymbol{d} \in D$ and taking the union, *i.e.* computing $\tau^{\otimes}(X)$ by $X \sqcup \bigsqcup_{\boldsymbol{d} \in D} X_{\boldsymbol{d}}$ with $X_{\boldsymbol{d}} = \big((X \sqcap G) \nearrow \{\boldsymbol{d}\}\big) \sqcap (G + \{\boldsymbol{d}\})$. However, there is a subtle difference: this formula computes the correct set for all states reachable within $G$, but for the last step crossing the border of $G$ it allows only those vectors $\boldsymbol{d}$ having been used for the previous iterations, whereas actually there is a choice among all $\boldsymbol{d} \in D$.

We combine Propositions 1 and 2 to formulate the following theorem:

**Theorem 3.** *The accelerated transition $\tau^{\otimes}$ for a translation with inputs and a simple guard*

$$\tau : (\mathbf{A}\boldsymbol{x} \leq \boldsymbol{b}) \wedge (\mathbf{J}\boldsymbol{\xi} \leq \boldsymbol{k}) \quad \rightarrow \quad \boldsymbol{x}' = \boldsymbol{x} + \mathbf{T}\boldsymbol{\xi} + \boldsymbol{u}$$

*can be computed by*

$$\tau^{\otimes}(X) = X \sqcup \tau\big((X \sqcap G) \nearrow D\big)$$

*where $G = (\mathbf{A}\boldsymbol{x} \leq \boldsymbol{b})$ and $D = \{\boldsymbol{d} \mid \exists \boldsymbol{\xi} : \boldsymbol{d} = \boldsymbol{T}\boldsymbol{\xi} + \boldsymbol{u} \wedge \boldsymbol{J}\boldsymbol{\xi} \leq \boldsymbol{k}\}$.*

**Example 1.** Consider the polyhedron $X = \{(x_1, x_2) \mid 0 \leq x_1 \leq x_2 \leq 1\}$ and the transition

$$\tau : \left| \begin{array}{c} x_1 + x_2 \ \leq \ 4 \\ 1 \leq \xi \ \leq \ 2 \end{array} \right. \rightarrow \left| \begin{array}{l} x_1' = x_1 + 2\xi - 1 \\ x_2' = x_2 + \xi \end{array} \right.$$

Eliminating the inputs as in Proposition 1 yields $D = \{(d_1, d_2) | 1 \leq d_1 \leq 3 \wedge -d_1 + 2d_2 = 1\}$, see Fig. 5 left-hand side. After translation of $X$ by $D$ (Fig. 5 right-hand side) we obtain the polyhedron $\{(x_1, x_2) | x_1 \geq 0 \wedge -x_1 + x_2 \leq 1 \wedge x_1 + x_2 \leq 9 \wedge -2x_1 + 4x_2 \leq 9 \wedge 2x_1 - 3x_2 \leq 0\}$.

**Remark 5.** In analogy to Theorem 1, we could alternatively consider the formula

$$X \sqcup ((((X \sqcap G) \nearrow D) \sqcap (G + D)).$$

In order to justify this, we extend the proof of Proposition 2 by continuing at the label (dense approximation):
$\Leftrightarrow \exists \alpha \geq 0, \exists \boldsymbol{x}_0 \in X \sqcap G, \exists \boldsymbol{x}_k, \exists \boldsymbol{d}, \boldsymbol{d}' \in D : \boldsymbol{x}' = \boldsymbol{x}_k + \boldsymbol{d}' \ \wedge \ \boldsymbol{x}_k = \boldsymbol{x}_0 + \alpha \boldsymbol{d} \ \wedge \ G(\boldsymbol{x}_k)$
$\Leftrightarrow \exists \alpha \geq 0, \exists \boldsymbol{x}_0 \in X \sqcap G, \exists \boldsymbol{d}, \boldsymbol{d}' \in D : \boldsymbol{x}' = \boldsymbol{x}_0 + \alpha \boldsymbol{d} + \boldsymbol{d}' \ \wedge \ G(\boldsymbol{x}' - \boldsymbol{d}')$
$\Rightarrow \big(\exists \alpha \geq 0, \exists \boldsymbol{x}_0 \in X \sqcap G, \exists \boldsymbol{d}, \boldsymbol{d}' \in D : \boldsymbol{x}' = \boldsymbol{x}_0 + \alpha \boldsymbol{d} + \boldsymbol{d}'\big) \ \wedge \ \big(\exists \boldsymbol{d}' \in D : G(\boldsymbol{x}' - \boldsymbol{d}')\big)$

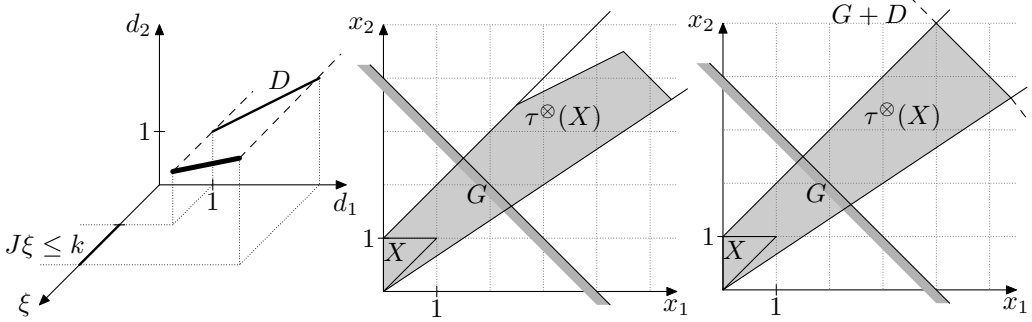


Fig. 5. Translation with inputs (Example 1): The left-hand side shows the transformation of the inputs: $\mathbf{J}\boldsymbol{\xi} \leq \boldsymbol{k} \wedge \boldsymbol{d} = \mathbf{T}\boldsymbol{\xi} + \boldsymbol{u}$ (bold line) is projected on variables $\boldsymbol{d}$. The shadowed area in the right-hand side figure is $\tau^{\otimes}(X)$.

Fig. 6. Precision loss in example 1 when using the approximate formula according to Remark 5.

$\Leftrightarrow \big(\exists \alpha' \geq 1, \exists \boldsymbol{x}_0 \in X \sqcap G, \exists \boldsymbol{d}'' \in D : \boldsymbol{x}' = \boldsymbol{x}_0 + \alpha' \boldsymbol{d}''\big) \ \wedge \ \big(\exists \boldsymbol{d}' \in D : G(\boldsymbol{x}' - \boldsymbol{d}')\big)$
$\Rightarrow \boldsymbol{x}' \in (X \sqcap G) \nearrow D \ \wedge \ \boldsymbol{x}' \in (G + D)$
using $\{\boldsymbol{x} \mid \exists \boldsymbol{d} \in D \wedge G(\boldsymbol{x} - \boldsymbol{d})\} = \{\boldsymbol{z} + \boldsymbol{d} \mid \boldsymbol{d} \in D \wedge G(\boldsymbol{z})\} = (G + D)$. But it can be observed that for the translation of example 1 the latter formula results in an over-approximation (see Fig. 6) as compared to the result in Fig. 5. This reflects the additional approximation steps in the proof.

### *4.2. Translations/Resets with inputs and simple guards*

Translations/resets with inputs and simple guards are defined by

$$\underbrace{\begin{pmatrix} \mathbf{A} & 0 \\ 0 & \mathbf{J} \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{\xi} \end{pmatrix} \leq \begin{pmatrix} \boldsymbol{b} \\ \boldsymbol{k} \end{pmatrix}}_{\mathbf{A}\boldsymbol{x} \leq \boldsymbol{b} \wedge \mathbf{J}\boldsymbol{\xi} \leq \boldsymbol{k}} \rightarrow \boldsymbol{x}' = \underbrace{\begin{pmatrix} \mathbf{C} & \mathbf{T} \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{\xi} \end{pmatrix} + \boldsymbol{u}}_{\mathbf{C}\boldsymbol{x} + \mathbf{T}\boldsymbol{\xi} + \boldsymbol{u}}$$

where $C$ is a diagonal matrix with $C_{i,i} \in \{0, 1\}$ for all $i$.

*Notations.* Let $\mathbf{C}' = \mathbf{I} - \mathbf{C}$ with $\mathbf{I}$ the identity matrix. Any vector $\boldsymbol{x}$ can be decomposed in $\boldsymbol{x} = \boldsymbol{x}^{t,0} + \boldsymbol{x}^{0,r}$ with $\boldsymbol{x}^{t,0} = \mathbf{C}\boldsymbol{x}$ and $\boldsymbol{x}^{0,r} = \mathbf{C}'\boldsymbol{x}$. We extend such notations to sets: $X^{t,0} = \{\boldsymbol{x}^{t,0} \mid \boldsymbol{x} \in X\}$ and $X^{0,r} = \{\boldsymbol{x}^{0,r} \mid \boldsymbol{x} \in X\}$. Instead of resetting dimensions, one can quantify them: $X^{t,\bullet} = \{\boldsymbol{x} \mid \boldsymbol{x}^{t,0} \in X^{t,0}\}$ and $X^{\bullet,r} = \{\boldsymbol{x} \mid \boldsymbol{x}^{0,r} \in X^{0,r}\}$ denote the set of vectors obtained by existential quantification of the reset (resp. translated) dimensions. $\boldsymbol{x}^t$ and $\boldsymbol{x}^r$ denote the projection of $\boldsymbol{x}$ on the subspace of translated (resp. reset) dimensions. $I$ denotes the set of dimensions, $I^t = \{i \in I \mid C_{i,i} = 1\}$ and $I^r = I \backslash I_t$ are the set of translated and reset dimensions.

Observe, that the over-approximation $X^{t,\bullet} \sqcap X^{\bullet,r}$ of a set $X$ (by the cartesian product w.r.t. to dimensions $I^t$ and $I^r$) is equal to the Minkowski sum $X^{t,0} + X^{0,r}$.

The case of translations/resets with inputs can be handled in a way similar to Section 4.1: we combine Proposition 1 and Theorem 2 to reduce translations/resets with inputs to *polyhedral translations with resets* denoted $\tau : G \rightarrow \boldsymbol{x}' = \mathbf{C}\boldsymbol{x} + D$ and with the meaning $\tau(X) = (X \sqcap G)^{t,0} + D$.

Mind, however, that remark 1 does not apply any more and cannot be exploited in the presence of inputs, because the variables being reset may be assigned a different value in each iteration.

**Proposition 3.** Let $\tau$ be a polyhedral translation with resets $G \to \boldsymbol{x}' = \mathbf{C}\boldsymbol{x} + D$. Then, the set

$$\tau^{\otimes}(X) = X \sqcup \tau(X) \sqcup \tau\Big(\big((\tau(X) \sqcap G)^{t,0} \nearrow D^{t,0}\big) + D^{0,r}\Big)$$

is a convex over-approximation of $\tau^*(X)$.

In the formula above and in the proof below, we unfold $\tau$ twice, that is, we accelerate only the central part of the sequence $\boldsymbol{x} \xrightarrow{\tau} \boldsymbol{x}_0 \dots \boldsymbol{x}_n \xrightarrow{\tau} \boldsymbol{x'}$ with $\boldsymbol{x} \in X$ because we have $\forall k \in [0, n] : \boldsymbol{x_k} \in G \sqcap D^{\bullet,r}$, whereas we only have $\boldsymbol{x} \in G$ at the start-point, and $\boldsymbol{x'} \in D^{\bullet,r}$ at the end-point.

**Proof.** The formula is trivially correct for 0 or 1 iterations of the self-loop $\tau$. It remains to show that for the case of $k \geq 2$ iterations our formula yields an over-approximation of $\bigcup_{k\geq 2} \tau^k(X)$.

$\boldsymbol{x}' \in \bigcup_{k\geq 2} \tau^k(X) \Leftrightarrow \boldsymbol{x}' \in \tau\left(\bigcup_{k\geq 0} \tau^k(\tau(X))\right)$

$\Leftrightarrow \exists k \geq 0, \exists \boldsymbol{x} \in X, \exists \boldsymbol{x}_0 \dots \boldsymbol{x}_k, \exists \boldsymbol{d}_1 \dots \boldsymbol{d}_k \in D$ :

$$\begin{cases} \quad \boldsymbol{x}_0 \in \tau(\boldsymbol{x}) \\ \wedge\ \forall k' \in [1,k] \ : \ \begin{cases} \boldsymbol{x}^i_{k'} = \boldsymbol{x}^i_0 + \sum_{j=1}^{k'} \boldsymbol{d}^i_j & \text{for } i \in I^t \\ \boldsymbol{x}^i_{k'} = \boldsymbol{d}^i_{k'} & \text{for } i \in I^r \end{cases} \\ \wedge\ \boldsymbol{x}' \in \tau(\boldsymbol{x}_k) \\ \wedge\ \forall k' \in [0,k] \ : \ G(\boldsymbol{x}_{k'}) \end{cases}$$

$\Leftrightarrow \exists k \geq 0, \exists \boldsymbol{x} \in X, \exists \boldsymbol{x}_0 \dots \boldsymbol{x}_k, \exists \boldsymbol{d}_1 \dots \boldsymbol{d}_k \in D$ :

$$\begin{cases} \quad \forall k' \in [1,k] \ : \ \boldsymbol{x}_{k'} = \boldsymbol{x}_0^{t,0} + (\sum_{j=1}^{k'} \boldsymbol{d}_j^{t,0}) + \boldsymbol{d}_{k'}^{0,r} \\ \wedge\ \forall k' \in [0,k] \ : \ G(\boldsymbol{x}_{k'}) \\ \wedge\ \boldsymbol{x}_0 \in \tau(\boldsymbol{x}) \ \wedge \ \boldsymbol{x}' \in \tau(\boldsymbol{x}_k) \end{cases}$$

$\Rightarrow \exists k \geq 0, \exists \boldsymbol{x} \in X, \exists \boldsymbol{x}_0 \dots \boldsymbol{x}_k, \exists \boldsymbol{d}_1^{t,0} \dots \boldsymbol{d}_k^{t,0} \in D^{t,0}, \exists \boldsymbol{d}_1^{0,r} \dots \boldsymbol{d}_k^{0,r} \in D^{0,r}$ :

$$\begin{cases} \quad \forall k' \in [1,k] \ : \ \boldsymbol{x}_{k'} = \boldsymbol{x}_0^{t,0} + (\sum_{j=1}^{k'} \boldsymbol{d}_j^{t,0}) + \boldsymbol{d}_{k'}^{0,r} \\ \wedge\ \forall k' \in [0,k] \ : \ G(\boldsymbol{x}_{k'}) \\ \wedge\ \boldsymbol{x}_0 \in \tau(\boldsymbol{x}) \ \wedge \ \boldsymbol{x}' \in \tau(\boldsymbol{x}_k) \end{cases}$$

($D$ approximated by the sum $(D^{t,0} + D^{0,r})$)

$\Leftrightarrow \exists k \geq 0, \exists \boldsymbol{x} \in X, \exists \boldsymbol{x}_0, \boldsymbol{x}_k, \exists \boldsymbol{d}^{t,0} \in D^{t,0}, \exists \boldsymbol{d}_k^{0,r} \in D^{0,r}$ :

$$\begin{cases} \quad \boldsymbol{x}_k = \boldsymbol{x}_0^{t,0} + k\boldsymbol{d}^{t,0} + \boldsymbol{d}_k^{0,r} \\ \wedge\ G(\boldsymbol{x}_0) \ \wedge \ G(\boldsymbol{x}_k) \\ \wedge\ \boldsymbol{x}_0 \in \tau(\boldsymbol{x}) \ \wedge \ \boldsymbol{x}' \in \tau(\boldsymbol{x}_k) \end{cases}$$

(because $D^{t,0}, D^{0,r}$ and $G$ are convex and $\boldsymbol{x}_0^{0,r} \in D^{0,r}$)

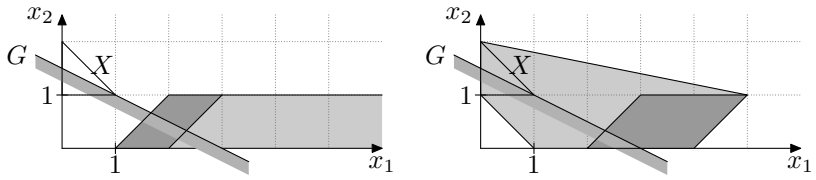


Fig. 7. Translation/reset with inputs: Example 2. Left-hand side: $\tau(X)$ (dark shadowed) and $((\tau(X) \sqcap G)^{t,0} \nearrow D^{t,0}) + D^{0,r}$ (whole shadowed area). Right-hand side: $\tau(((\tau(X) \sqcap G)^{t,0} \nearrow D^{t,0}) + D^{0,r})$ (dark shadowed) and $\tau^{\otimes}(X)$ (whole shadowed area).

$\Rightarrow \exists \alpha \geq 0, \exists \boldsymbol{x} \in X, \exists \boldsymbol{x}_0, \boldsymbol{x}_k, \exists \boldsymbol{d}^{t,0} \in D^{t,0}, \exists \boldsymbol{d}_k^{0,r} \in D^{0,r}$ :

$$\begin{cases} \wedge\ \boldsymbol{x}_k = \boldsymbol{x}_0^{t,0} + \alpha \boldsymbol{d}^{t,0} + \boldsymbol{d}_k^{0,r} \\ \wedge\ \boldsymbol{x}_0 \in \tau(\boldsymbol{x})\ \wedge\ G(\boldsymbol{x}_0)\ \wedge\ \boldsymbol{x}' \in \tau(\boldsymbol{x}_k) \end{cases}$$

(dense over-approximation; $G(\boldsymbol{x}_k)$ already implied by $\boldsymbol{x}' \in \tau(\boldsymbol{x}_k)$)

$\Leftrightarrow \boldsymbol{x}' \in \tau\Big(\big((\tau(X) \sqcap G)^{t,0} \nearrow D^{t,0}\big) + D^{0,r}\Big)$

□

**Theorem 4.** *The accelerated transition $\tau^{\otimes}$ for a translation/reset with inputs and a simple guard $\tau$ can be computed by applying Proposition 3 with $D$ defined as in Proposition 1.*

**Example 2.** Consider the polyhedron $X = \{(x_1, x_2) \mid 0 \leq x_1 \wedge 1 \leq x_2 \wedge x_1 + x_2 \leq 2\}$ and the transition $\tau : \left| \begin{array}{r} x_1 + 2x_2 \leq 3 \\ 0 \leq \xi \leq 1 \end{array} \right. \rightarrow \left| \begin{array}{l} x_1' = x_1 + \xi + 1 \\ x_2' = \xi \end{array} \right.$
Eliminating the inputs yields $D = \{(d_1, d_2) \mid 1 \leq d_1 \leq 2 \wedge d_1 - d_2 = 1\}$ and $D^{t,0} = \{(d_1, d_2) \mid 1 \leq d_1 \leq 2 \wedge d_2 = 0\}$. We obtain $\tau^{\otimes}(X) = \{(x_1, x_2) \mid x_1 + x_2 \geq 1 \wedge x_2 \geq 0 \wedge x_1 - x_2 \leq 4 \wedge x_1 + 5x_2 \leq 10 \wedge x_1 \geq 0\}$, see Fig. 7.

### *4.3. Weakening general guards to simple guards*

As discussed at the beginning of Section 4, allowing constraints that realte state variables with input variables in guards, *i.e.* $G = \mathbf{A}\boldsymbol{x} + \mathbf{L}\boldsymbol{\xi} \leq \boldsymbol{b} \wedge \mathbf{J}\boldsymbol{\xi} \leq \boldsymbol{k}$ with $\mathbf{L} \neq 0$ (see Eqn. (5)), makes acceleration very difficult. Our solution is to weaken the guard $G$ by the simple guard (or cartesian product) $\overline{G} = \underbrace{(\exists \boldsymbol{\xi} : G)}_{\mathbf{A}'\boldsymbol{x} \leq \boldsymbol{b}'} \wedge \underbrace{(\exists \boldsymbol{x} : G)}_{\mathbf{J}'\boldsymbol{\xi} \leq \boldsymbol{k}'}$.

In the case of a translation $\tau : G \rightarrow \boldsymbol{x}' = \boldsymbol{x} + \mathbf{T}\boldsymbol{\xi} + \boldsymbol{u}$, we can now apply the accelerated transition from Theorem 3

$$\tau^{\otimes}(X) = X \sqcup \tau\big((X \sqcap G') \nearrow D'\big)$$

with $G' = (\mathbf{A}'\boldsymbol{x} \leq \boldsymbol{b}')$ and $D' = \{\boldsymbol{d} \mid \exists \boldsymbol{\xi} : \boldsymbol{d} = \boldsymbol{T}\boldsymbol{\xi} + \boldsymbol{u} \wedge \boldsymbol{J}'\boldsymbol{\xi} \leq \boldsymbol{k}'\}$. This trivially results in a sound over-approximation because a weaker guard is used for abstract acceleration. For translations with resets, Theorem 4 can be applied analogously.

Notice however that in those theorems, we can still compute exactly the function $\tau$ using the original guard $G$. Indeed, the proofs of those theorems are not based on the assumption $\mathbf{L} \neq 0$ when they introduce the function $\tau$.

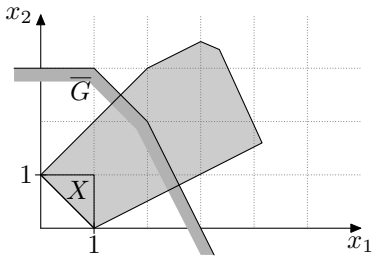


Fig. 8. Example 3: accelerated transition $\tau^{\otimes}(X)$ using the weakened guard $\overline{G}$ (result shadowed).

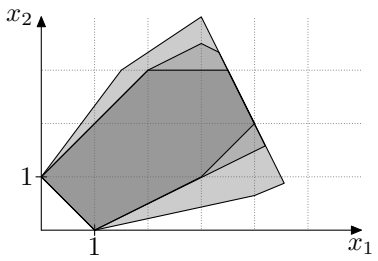


Fig. 9. Example 3: comparison between convex hull of the exact result (dark gray), our method (gray), and widening with no delay and 3 (!) descending iterations (light gray).

**Example 3.** Consider the polyhedron $X = \{(x_1, x_2) \mid x_1 \leq 1 \wedge x_2 \leq 1 \wedge x_1 + x_2 \geq 1\}$ and the transition $\tau : \left| \begin{array}{r} 2x_1 + x_2 + \xi \leq 6 \\ x_2 - \xi \leq 2 \\ 0 \leq \xi \leq 1 \end{array} \right| \rightarrow \left| \begin{array}{l} x_1' = x_1 + \xi + 1 \\ x_2' = x_2 + 1 \end{array} \right.$

The weakened guard is $\overline{G} = (2x_1 + x_2 \leq 6 \wedge x_1 + x_2 \leq 4 \wedge x_2 \leq 3) \wedge (0 \leq \xi \leq 1)$. Eliminating the inputs yields $D = \{(d_1, d_2) \mid 1 \leq d_1 \leq 2 \wedge d_2 = 1\}$. We obtain $\tau^{\otimes}(X) = \{(x_1, x_2) \mid x_1 + x_2 \geq 1 \wedge x_2 - x_1 \leq 1 \wedge -4 \leq x_1 - 2x_2 \leq 1 \wedge x_1 + 2x_2 \leq 10 \wedge 2x_1 + x_2 \leq 10\}$, see Fig. 8. The convex hull of the exact result is $\{(x_1, x_2) | x_1 + x_2 \geq 1 \wedge -2 \leq x_2 - x_1 \leq 1 \wedge x_1 - 2x_2 \leq 1 \wedge x_2 \leq 3 \wedge 2x_1 + x_2 \leq 10\}$, see Fig. 9.

## 5. Backward Acceleration

Acceleration has been applied to forward reachability analysis in order to compute the reachable states starting from a set of initial states. In the verification of safety properties it is also useful to compute the set of states that are co-reachable starting from those *final* states (also often referred to as "bad" or unsafe states) that do not fulfill the property. If the co-reachable set obtained by executing the program backwards does not intersect with the set of initial states then the program is safe. Such a backward co-reachability analysis has other applications: for example, it allows to *synthesize* constraints on *parameter* variables that ensure that a property is satisfied (see *e.g.* Alur et al. (1995)). It can also be combined by intersection with a forward analysis, so as to obtain an approximation of the sets of states belonging to a path from initial to final states (see for instance Jeannet (2003)).

Again, we present how to compute the accelerated transitions in the case of translations and translations with resets. Although the inverse of a translation is a translation, the difference is that the intersection with the guard occurs after the (inverted) translation. The case of translations with resets is more complicated than for the forward case.

### *5.1. Translations*

**Proposition 4.** Let $\tau$ be a polyhedral translation $G \rightarrow \boldsymbol{x'} = \boldsymbol{x} + D$. Then the set

$$\tau^{-\otimes}(X') = X' \sqcup \big((\tau^{-1}(X') \nearrow (-D)\big) \sqcap G$$

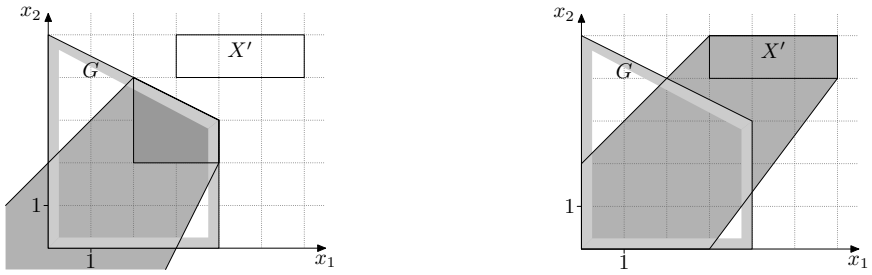


Fig. 10. Backward acceleration of a translation loop (Ex. 4) starting from $X'$ with $\tau^{-1}(X')$ (dark shadowed) and $\tau^{-1}(X') \nearrow (-D)$ (whole shadowed area) on the left-hand side and the final result (right-hand side).

is a convex over-approximation of $\tau^{-*}(X')$, where $\tau^{-*} = (\tau^{-1})^* = (\tau^*)^{-1}$ is the reflexive and transitive backward closure of $\tau$.

The negation of a polyhedron has the following meaning: $\boldsymbol{d} \in (-D) \Leftrightarrow (-\boldsymbol{d}) \in D$

**Proof.**

$\boldsymbol{x}_0 \in \bigcup_{k\geq 1} \tau^{-k}(X') \Leftrightarrow \exists \boldsymbol{x}' \in X' : \boldsymbol{x}' \in \tau\left(\bigcup_{k\geq 0} \tau^k(\{\boldsymbol{x}_0\})\right)$

$$\Leftrightarrow \exists k \geq 0, \exists \boldsymbol{x}' \in X', \exists \boldsymbol{x}_k, \exists \boldsymbol{d}_1, \ldots, \boldsymbol{d}_k \in D \ : \begin{cases} \boldsymbol{x}_k = \boldsymbol{x}_0 + \sum_{j=1}^{k} \boldsymbol{d}_j \\ \forall k' \in [0,k] \ : \ G(\boldsymbol{x}_0 + \sum_{j=1}^{k'} \boldsymbol{d}_j) \\ \boldsymbol{x}' \in \tau(\{\boldsymbol{x}_k\}) \end{cases}$$

(forward reachability)

$$\Leftrightarrow \exists k \geq 0, \exists \boldsymbol{x}' \in X', \exists \boldsymbol{x}_k, \exists \boldsymbol{d}_1, \ldots, \boldsymbol{d}_k \in D \ : \begin{cases} \boldsymbol{x}_k \in \tau^{-1}(\{\boldsymbol{x}'\}) \\ \boldsymbol{x}_0 = \boldsymbol{x}_k - \sum_{j=1}^{k} \boldsymbol{d}_j \\ \forall k' \in [0,k] \ : \ G(\boldsymbol{x}_k - \sum_{j=k'+1}^{k} \boldsymbol{d}_j) \end{cases}$$

(rewritten as backward reachability)

$$\Leftrightarrow \exists k \geq 0, \exists \boldsymbol{x}' \in X', \exists \boldsymbol{x}_k, \exists \boldsymbol{d} \in D \ : \ \boldsymbol{x}_k \in \tau^{-1}(\{\boldsymbol{x}'\}) \ \wedge \ \boldsymbol{x}_0 = \boldsymbol{x}_k - k\boldsymbol{d} \ \wedge \ G(\boldsymbol{x}_0) \ \wedge \ G(\boldsymbol{x}_k)$$

(because $D$ and $G$ are convex)

$$\Rightarrow \exists \alpha \geq 0, \exists \boldsymbol{x}' \in X', \exists \boldsymbol{x}_k, \exists \boldsymbol{d} \in D \ : \ \boldsymbol{x}_k \in \tau^{-1}(\{\boldsymbol{x}'\}) \ \wedge \ \boldsymbol{x}_0 = \boldsymbol{x}_k - \alpha\boldsymbol{d} \ \wedge \ G(\boldsymbol{x}_0)$$

(dense approximation; $G(\boldsymbol{x}_k)$ implied by $\boldsymbol{x}_k \in \tau^{-1}(\{\boldsymbol{x}'\})$)

$\Leftrightarrow \boldsymbol{x}_0 \in \left((\tau^{-1}(X') \nearrow (-D)\right) \sqcap G.$

□

**Example 4.** Consider the polyhedron $X' = \{(x_1, x_2) \mid 3 \leq x_1 \leq 6 \ \wedge \ 4 \leq x_2 \leq 5\}$ and the transition $\tau : \left| \begin{array}{c} x_1 + 2x_2 \leq 10 \ \wedge 0 \leq x_1 \leq 4 \ \wedge \\ 0 \leq x_2 \ \wedge 1 \leq \xi \leq 2 \end{array} \right| \rightarrow \left| \begin{array}{l} x_1' = x_1 + 1 \\ x_2' = x_2 + \xi \end{array} \right.$

The polyhedron $D$ is $\{(d_1, d_2) \,|\, d_1 = 1 \wedge 1 \leq d_2 \leq 2\}$. As result of the backward acceleration (Fig. 10) we obtain the polyhedron $\{(x_1, x_2) \mid 0 \leq x_1 \leq 6 \ \wedge \ 0 \leq x_2 \leq 5 \ \wedge \ -x_1 + x_2 \leq 2 \ \wedge \ 4x_1 - 3x_2 \leq 12\}$.

*5.2. Translations/resets*

**Proposition 5.** . Let $\tau$ be a polyhedral translation with resets $G \to \boldsymbol{x}' = \mathbf{C}\boldsymbol{x} + D$. Then, the set

$$\tau^{-\otimes}(X') = X' \sqcup \tau^{-1}(X') \sqcup \tau^{-1}\Big(\big((\tau^{-1}(X')^{t,\bullet} \nearrow (-D^{t,0})\big) \sqcap G\Big)$$

is a convex over-approximation of $\tau^{-*}(X')$.

**Proof.** The formula is trivially correct for 0 or 1 backward iterations of the self-loop $\tau$, so, it remains to show that for the case of $k \geq 2$ iterations our formula yields an over-approximation of $\bigcup_{k\geq 2} \tau^{-k}(X)$.

$\boldsymbol{x} \in \bigcup_{k\geq 2} \tau^{-k}(X') \Leftrightarrow \exists \boldsymbol{x'} \in X' \ : \ \boldsymbol{x}' \in \tau\left(\bigcup_{k\geq 0} \tau^k(\tau(\boldsymbol{x}))\right)$

$\Leftrightarrow \exists k \geq 0, \exists \boldsymbol{x}' \in X', \exists \boldsymbol{x}_0 \dots \boldsymbol{x}_k, \exists \boldsymbol{d}_1 \dots \boldsymbol{d}_k \in D \ :$

$$\begin{cases} \quad \boldsymbol{x}_0 \in \tau(\boldsymbol{x}) \wedge \boldsymbol{x}' \in \tau(\boldsymbol{x}_k) \\ \wedge \ \forall k' \in [1,k] \ : \ \boldsymbol{x}_{k'} = \boldsymbol{x}_0^{t,0} + \sum_{j=1}^{k'} \boldsymbol{d}_j^{t,0} + \boldsymbol{d}_{k'}^{0,r} \\ \wedge \ \forall k' \in [0,k] \ : \ G(\boldsymbol{x}_{k'}) \end{cases}$$

(forward reachability)

$\Leftrightarrow \exists k \geq 0, \exists \boldsymbol{x}' \in X', \exists \boldsymbol{x}_0 \dots \boldsymbol{x}_k, \exists \boldsymbol{d}_1 \dots \boldsymbol{d}_k \in D \ :$

$$\begin{cases} \quad \forall k' \in [0,k-1] \ : \ \boldsymbol{x}_{k'} = \boldsymbol{x}_k^{t,0} - \sum_{j=k'+1}^{k} \boldsymbol{d}_j^{t,0} + \boldsymbol{d}_{k'}^{0,r} \\ \wedge \ \forall k' \in [0,k] \ : \ G(\boldsymbol{x}_{k'}) \\ \wedge \ \boldsymbol{x}_k \in \tau^{-1}(\{\boldsymbol{x}'\}) \ \wedge \ \boldsymbol{x} \in \tau^{-1}(\{\boldsymbol{x}_0\}) \end{cases}$$

(rewritten as backward reachability)

$\Rightarrow \exists k \geq 0, \exists \boldsymbol{x}' \in X', \exists \boldsymbol{x}_0 \dots \boldsymbol{x}_k, \exists \boldsymbol{d}_1^{t,0} \dots \boldsymbol{d}_k^{t,0} \in D^{t,0}, \exists \boldsymbol{d}_1^{0,r} \dots \boldsymbol{d}_k^{0,r} \in D^{0,r} \ :$

$$\begin{cases} \quad \forall k' \in [0,k-1] \ : \ \boldsymbol{x}_{k'} = \boldsymbol{x}_k^{t,0} - \sum_{j=k'+1}^{k} \boldsymbol{d}_j^{t,0} + \boldsymbol{d}_{k'}^{0,r} \\ \wedge \ \forall k' \in [0,k] \ : \ G(\boldsymbol{x}_{k'}) \\ \wedge \ \boldsymbol{x}_k \in \tau^{-1}(\{\boldsymbol{x}'\}) \ \wedge \ \boldsymbol{x} \in \tau^{-1}(\{\boldsymbol{x}_0\}) \end{cases}$$

($D$ approximated by the sum $(D^{t,0} + D^{0,r})$)

$\Leftrightarrow \exists k \geq 0, \exists \boldsymbol{x}' \in X', \exists \boldsymbol{x}_0, \boldsymbol{x}_k, \exists \boldsymbol{d}^{t,0} \in D^{t,0}, \exists \boldsymbol{d}^{0,r} \in D^{0,r} \ :$

$$\begin{cases} \quad \boldsymbol{x}_0 = \boldsymbol{x}_k^{t,0} - k\boldsymbol{d}^{t,0} + \boldsymbol{d}^{0,r} \\ \wedge \ G(\boldsymbol{x}_0) \wedge G(\boldsymbol{x}_k) \\ \wedge \ \boldsymbol{x}_k \in \tau^{-1}(\{\boldsymbol{x}'\}) \ \wedge \ \boldsymbol{x} \in \tau^{-1}(\{\boldsymbol{x}_0\}) \end{cases}$$

(because $D^{t,0}, D^{0,r}$ and $G$ are convex and $\boldsymbol{x}_k^{0,r} \in D^{0,r}$)

$\Rightarrow \exists \alpha \geq 0, \exists \boldsymbol{x}' \in X', \exists \boldsymbol{x}_0, \boldsymbol{x}_k, \exists \boldsymbol{d}^{t,0} \in D^{t,0}, \exists \boldsymbol{d}^{0,r} \in D^{0,r} \ :$

$$\begin{cases} \quad \boldsymbol{x}_0 = \boldsymbol{x}_k^{t,0} - \alpha\boldsymbol{d}^{t,0} + \boldsymbol{d}^{0,r} \\ \wedge \ \boldsymbol{x}_k \in \tau^{-1}(\{\boldsymbol{x}'\}) \ \wedge \ \boldsymbol{x} \in \tau^{-1}(\{\boldsymbol{x}_0\}) \ \wedge \ G(\boldsymbol{x}_0) \end{cases}$$

(dense approximation; $G(\boldsymbol{x}_k)$ implied by $\boldsymbol{x}_k \in \tau^{-1}(\{\boldsymbol{x}'\})$)

$\Leftrightarrow \boldsymbol{x} \in \tau^{-1}\Big(\big((\tau^{-1}(X'))^{t,\bullet} \nearrow (-D^{t,0})\big) \sqcap G\Big)$

(because $\boldsymbol{x} \in \tau^{-1}(\{\boldsymbol{x}'\}) \Rightarrow \boldsymbol{x}' \in D^{\bullet,r}$)

□

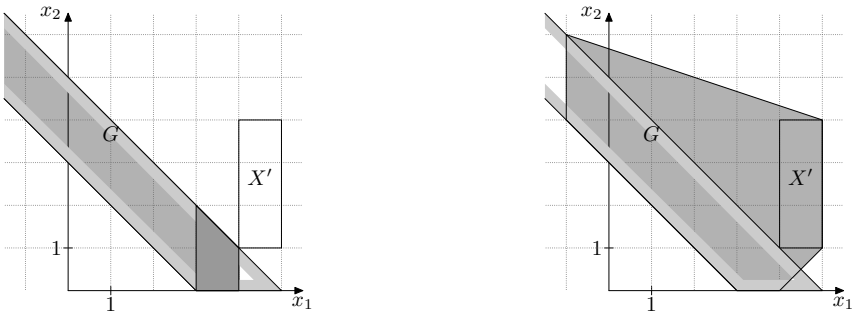


Fig. 11. Backward acceleration of a loop with translations and resets (Ex. 5) starting from the initial set $X'$. Right-hand side: $\tau^{-1}(X')$ (dark shadowed) and $\big((\tau^{-1}(X')|_t \nearrow (-D^t)\big) \sqcap G$ (whole shadowed area). Left-hand side: final result.

**Example 5.** Consider the polyhedron $X' = \{(x_1, x_2) \mid 4 \leq x_1 \leq 5 \ \wedge \ 1 \leq x_2 \leq 4\}$ and the transition $\tau : \left| \begin{array}{l} 3 \leq x_1 + x_2 \leq 5 \ \wedge \\ \qquad 1 \leq \xi \leq 3 \ \wedge 0 \leq x_2 \end{array} \right. \rightarrow \left| \begin{array}{l} x_1' = x_1 + 1 \\ x_2' = \xi \end{array} \right.$
The polyhedron $D$ is $\{(d_1, d_2) | d_1 = 1 \wedge 1 \leq d_2 \leq 3\}$. As result of the backward acceleration (Fig. 11) we obtain the polyhedron $\{(x_1, x_2) | -1 \leq x_1 \leq 5 \wedge 0 \leq x_2 \wedge x_1 + x_2 \geq 3 \wedge x_1 - x_2 \leq 4 \ \wedge \ x_1 + 3x_2 \leq 17\}$.

## 6. Evaluation

In the light of the restriction of abstract acceleration to some frequently occurring transition types (as exposed in Sec. 3), this section explains the advantages of this method in comparison with more general abstract-interpretation-based methods like standard widening (Cousot and Cousot (1977)) and the affine derivative closure method of Ancourt et al. (2010).

### *6.1. Comparing abstract acceleration to Kleene iteration*

The classical method for computing the smallest fixed point of $X = I \sqcup \tau(X)$ is *Kleene iteration* and proceeds as follows:

$$X_0 = I \qquad X_1 = I \sqcup \tau(X_0) = I \sqcup \tau(I) \qquad X_2 = I \sqcup \tau(X_1) = I \sqcup \tau(I \sqcup \tau(I)) \qquad \ldots$$

In contrast, abstract acceleration computes an over-approximation of the limit of the sequence

$$X_0' = I \qquad X_1' = I \sqcup \tau(I) \qquad X_2' = I \sqcup \tau(I) \sqcup \tau^2(I) \qquad \ldots$$

Abstract acceleration is *more precise* than Kleene iteration (assuming convergence), because in general $\tau$ does not distribute over $\sqcup$ and satisfies only the weaker property $\tau(X_1) \sqcup \tau(X_2) \sqsubseteq \tau(X_1 \sqcup X_2)$. For instance, if $I = [0, 0]$ and $\tau : x \leq 1 \rightarrow x' = x + 2$, we have $X_2 = [0, 0] \sqcup \tau([0, 2]) = [0, 3]$ and $X_2' = [0, 0] \sqcup [2, 2] \sqcup \bot = [0, 2]$. This observation corresponds to the debate about *Minimal-Fixed-Point* (MFP) *vs. Merge-Over-All-Paths* (MOP) solutions by Kam and Ullman (1977). Here, Kleene iteration corresponds to the MFP formulation and abstract acceleration to MOP.

Fig. 12 shows an example illustrating this issue: Kleene iteration translates the approximation added by the convex union with the result of the previous iteration in each

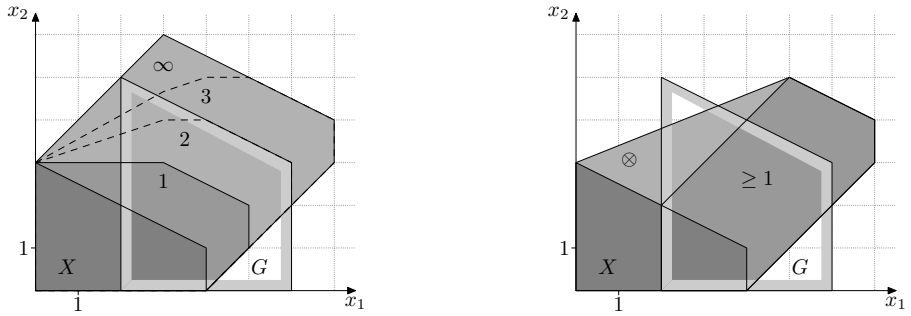


Fig. 12. Comparison between Kleene iteration (left-hand side: $X$ dark gray, iteration 1 medium gray, iterations 2 and 3 dashed, final result whole shadowed area) and abstract acceleration (right-hand side: iterations $\geq 1$ medium gray, final result whole shadowed area). $\tau$ is the translation $G \to (x_1', x_2') = (x_1+1, x_2+1)$.

step and converges slowly. Abstract acceleration translates only the intersection with the guard and takes the union as a last step.

The same effect occurs in backward acceleration.

### *6.2. Comparing abstract acceleration to widening*

The standard widening operator for convex polyhedra and refinements of it like limited widening (Halbwachs et al. (1997)) [1] may sometimes lead to good results. In this section, we compare the acceleration and the widening approaches on Examples 2 and 3. Analyzing such a program using widening after a number $N$ of initial steps resorts to computing the limit of the sequences

$$\begin{aligned} Y_0 &= X & Z_0 &= Y_N \\ Y_{n+1} &= X \sqcup \tau(Y_n) \quad \text{for } n < N & Z_{n+1} &= Z_n \nabla (Z_n \sqcup \tau(Z_n)) \end{aligned}$$

in which $X_n, Y_n, Z_n$ are associated with location $l_1$ on Fig. 13 (right-hand side). The properties of the widening operator $\nabla$ guarantee that the sequence $(Z_n)_{n\geq 0}$ converges in a finite number of steps to $Z_\infty$ (Cousot and Cousot (1992a)), which is an over-approximation of the reachable valuations at location $l_1$. This result may be improved by computing the first elements of the *narrowing* sequence $W_0 = Z_\infty, W_{n+1} = X \sqcup \tau(W_n)$, which does not necessarily converge.

We take again a look at the example in Fig. 12 (left-hand side) which depicts a typical situation where the widening operator loses precision: the constraints which make up the upper boundary of the polyhedron are shifted and rotated in each iteration. The extrapolation performed by standard widening simply removes such constraints. A descending iteration finally yields the same result as shown in Fig. 12 (left-hand side), which is less precise than the result obtained by acceleration shown in Fig. 12 (right-hand side).

[1] Limited widening is also called *widening with thresholds* (Cousot and Cousot (1992b)).

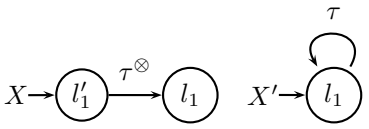


Fig. 13. Analysis with acceleration (left-hand side) and with widening (right-hand side) for Examples 2 and 3

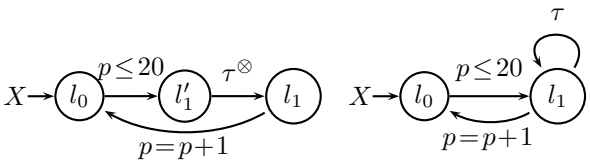


Fig. 14. Analysis with acceleration (left-hand side) and with widening (right-hand side) for Example 6.

*Translation/reset with inputs and simple guard.* If we compute the sequences defined above in the context of Example 2, we obtain with $N = 0$

$$Z_\infty = Z_1 = \{(x_1, x_2) \mid x_1 \geq 0\}$$

$$W_1 = \{x_1 \geq 0 \wedge x_2 \geq 1 \wedge x_1 + x_2 \geq 1 \wedge x_2 \leq 2\} \qquad W_\infty = W_2 = \tau^{\otimes}(X)$$

Delaying widening by one step ($N = 1$) improves the result for $Z_\infty$ and makes the sequence $(W_n)_{n\geq 0}$ converge in only one step:

$$Z_\infty = Z_1 = \{x_1 \geq 0 \wedge x_2 \geq 1 \wedge x_1 + x_2 \geq 1\}$$

$$W_\infty = W_1 = \tau^{\otimes}(X)$$

In both cases $Z_\infty$ is clearly much less precise than the result obtained by acceleration: neither $x_1$ nor $x_2$ have an upper bound (to be compared with Fig. 7).

One or two descending iterations allow to get the same result as the one obtained by acceleration. However, it should be pointed out that if this loop is a program fragment, for instance embedded in an outer loop as in Fig. 14, *it is not possible any more* to apply a descending iteration in the middle of an ascending iteration (otherwise convergence is not guaranteed). Moreover, the acceleration technique is more efficient computationally (in particular it does not require convergence tests), and it has a monotonic behavior, which is not the case of widening (see Bagnara et al. (2003)).

**Example 6.** To illustrate these points, we consider the program depicted on Fig. 14 in which the inner loop $\tau$ is adapted from Example 2:

$$\tau : \left| \begin{array}{r} 2x_1 + 2x_2 \leq p \\ 0 \leq y \leq 1 \end{array} \right. \rightarrow \left| \begin{array}{l} x'_1 = x_1 + y + 1 \\ x'_2 = y \\ p' = p \end{array} \right. , \; X = \left\{ (x_1, x_2, p) \left| \begin{array}{l} 0 \leq x_1 \;\wedge\; 1 \leq x_2 \\ x_1 + x_2 \leq 2 \wedge p = 3 \end{array} \right. \right\}$$

In both cases we apply widening on location $l_1$ with a delay $N \geq 1$, and we perform at least one descending iteration after convergence of the ascending iteration. Without acceleration, we obtain a very weak invariant:

$$Z_\infty = \{(x_1, x_2, p) \mid 0 \leq x_1 \wedge 3 \leq p\} \quad W_\infty = W_1 = \{(x_1, x_2, p) \mid 0 \leq x_1 \wedge 1 \leq x_1 + x_2 \wedge 3 \leq p\}$$

With acceleration we obtain much better results. We give here a simplified (over-approximated) invariant, because the actual result consists of more constraints:

$$W'_\infty = \{(x_1, x_2, p) \mid 0 \leq x_1 \leq 12 \wedge 0 \leq x_2 \leq 3 \wedge 3 \leq p \leq 20\}$$

One can also consider widening with thresholds, that keeps in the result of the widening operation the subset of a fixed set of *threshold constraints* that are satisfied by both of its arguments. In the case of Example 6, a natural threshold constraint set is defined by the postcondition of the guard of $\tau$ by the body of $\tau$, which is just $\tau(\top) = \{(x_1, x_2) \mid 0 \leq x_2 \leq$

1}. Using it with $N = 0$ one obtains the same $Z_\infty$ as with standard widening applied with $N = 1$. On Example 6 and with the same threshold set extended with $\{p \leq 21\}$, the results are improved but are still less precise than those obtained by combining acceleration and widening (in particular the descending iteration does not converge).

*Translation with inputs and non-simple guard.* In the context of Example 3, we obtain with $N = 0$:

$$\begin{aligned} Z_\infty = Z_1 &= \{(x_1, x_2) \mid x_1 + x_2 \geq 1\} \\ W_1 &= \{(x_1, x_2) \mid x_1 + x_2 \geq 1 \wedge 2x_1 + x_2 \leq 10 \wedge x_2 \leq 4 \wedge \\ &\qquad 0 \leq x_1 \leq 6 \wedge 3x1 + 5x_2 \geq 3\} \\ &\dots \\ W_3 &= \{(x_1, x_2) \mid x_1 + x_2 \geq 1 \wedge 2x_1 + x_2 \leq 10 \wedge 3x_2 - 2x_1 \leq 6 \wedge 3x_2 - 4x_1 \leq 3 \wedge \\ &\qquad 5x_1 - 22x_2 \leq 8 \wedge 29x_1 - 157x_2 \leq 29\} \sqsupset \tau^{\otimes}(X) \end{aligned}$$

Again $Z_\infty$ is very unprecise, but here the descending iteration does not converge (even if we use widening with thresholds), see Fig. 9 for $W_3$. If we use $N = 1$, then $Z_\infty$ is more precise, and $W_\infty = W_1 = \tau^{\otimes}(X)$.

These results are just small experiments, but they illustrate the sensitiveness of widening (if we delay it, it might improve the result, but this is not guaranteed either because it is not monotonic) and the fact that if the loop is part of a more complex program, the result might be much less precise, although more expensive to compute (delaying widening and applying descending iterations obviously increase the number of iterations).

### *6.3. Comparing abstract acceleration to derivative closure method*

The idea of the affine derivative closure algorithm (Ancourt et al. (2010)) is to compute an abstract transformer, *i.e.* a relation between variables $\boldsymbol{x}$ and $\boldsymbol{x'}$, independently of the initial state of the system. It *abstracts* the effect of the loop by a polyhedral translation

$$\mathit{true} \rightarrow \boldsymbol{x'} = \boldsymbol{x} + D_R \quad \text{with} \quad D_R = \{\boldsymbol{d} \,|\exists \boldsymbol{x}, \boldsymbol{x'} : R(\boldsymbol{x}, \boldsymbol{x'}) \wedge \boldsymbol{x'} = \boldsymbol{x} + \boldsymbol{d}\}$$

where $R$ captures the exact effect of one iteration of the loop. The polyhedron $D_R$ is called the "derivative" of the relation $R$. The effect of several self-loops represented by relations $R_1, \dots, R_k$ is abstracted by considering the convex union $\bigsqcup_i D_{R_i}$.

For translations, the method is equivalent to abstract acceleration and will yield the same results. On the contrary, resets cannot be expressed as polyhedral translations without losing information; for instance, if $R(x, x') = (x' = 0)$, then $D_R = \{d \,|\exists x, x' : x' = 0 \wedge x' = x + d\} = \top$. But for some examples, *e.g.* the car example in Fig. 2 – because the reset variable is related via the guards in both loops – it works well despite the presence of a reset and yields a precise invariant.

We illustrate the shortcomings of the technique in the following example that involves resets and inputs.

**Example 7.** (see Fig. 15)

$$\tau : \left| \begin{array}{r} x_1 \leq 4 \\ x_2 \leq 4 \\ 0 \leq \xi \leq 1 \end{array} \right. \rightarrow \left| \begin{array}{l} x_1' = x_1 + \xi + 1 \\ x_2' = \xi + 2 \end{array} \right. , \; X = \left\{ (x_1, x_2) \left| \begin{array}{l} x_1 \leq 1 \;\wedge\; x_2 \leq 1 \\ x_1 + x_2 \geq 1 \end{array} \right. \right\}$$

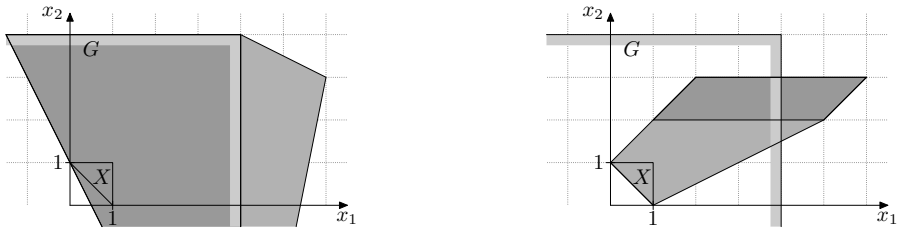


Fig. 15. Comparison between the derivative closure method (left-hand side: transitive closure intersected with the guard dark gray, final result whole shadowed area) and abstract acceleration (right-hand side: iterations $\geq 1$ dark gray, final result whole shadowed area) for Example 7.

The transformer of the loop

$$R(x_1, x_2, x_1', x_2', \xi) = \{x_1' = x_1 + \xi + 1 \wedge x_2' = \xi + 2 \wedge x_1 \leq 4 \wedge x_2 \leq 4 \wedge 0 \leq \xi \leq 1\}$$

expressed in terms of derivatives yields

$$\begin{aligned} D_R(d_1, d_2) &= \{\exists x_1, x_2, x_1', x_2', \xi : T(x_1, x_2, x_1', x_2', \xi) \wedge x_1' = x_1 + d_1 \wedge x_2' = x_2 + d_2\} \\ &= \{(d_1, d_2) \,|\, 1 \leq d_1 \leq 2 \wedge d_1 - d_2 \leq 3\} \end{aligned}$$

The closure of the loop starting from $X$

$$R^*(x_1', x_2') = \{\exists k \geq 0, x_1, x_2 : x_1' \geq x_1 + k \wedge x_1' \leq x_1 + 2k \wedge x_1' - x_2' \leq x_1 - x_2 + 3k \wedge X(x_1, x_2)\}$$

gives after intersection with the guard

$$R^*(x_1', x_2') \sqcap G(x_1', x_2') = \{(x_1', x_2') \,|\, 0 \leq x_1' \leq 4 \wedge 2x_1' + x_2' \geq 1 \wedge x_2' \leq 4\}$$

The final result is $\{(x_1', x_2') \,|\, x_2' \leq 4 \wedge 2x_1' + x_2' \geq 1 \wedge 5x_1' - x_2' \leq 27 \wedge x_1' + 2x_2' \leq 12\}$

Even though this method is only precise in the case of translations, its main advantage is that its application is more general than abstract acceleration, because it automatically approximates any kind of transitions that are not translations. Moreover, it can easily deal with multiple self-loops, whereas abstract acceleration requires more complicated graph transformations (see Gonnord (2007)).

## 7. Applying Abstract Acceleration

*Control flow graphs.* Program analysis using abstract interpretation in general and abstract acceleration in particular, is done on a control flow graph (CFG) of the program, which manipulates only numerical variables. This graph can be easily obtained from imperative programs by associating control points to programming constructs.

As mentioned in the introduction our goal is to analyze logico-numerical data-flow programs such as Lustre (Caspi et al. (1987)) programs: In order to reduce such a program to a purely numerical CFG, all possible valuations of Boolean state variables need to be *enumerated* and encoded in locations of the CFG. This *partitioning* and *partial evaluation* process may lead to a combinatorial explosion of control locations.

Jeannet (2003) presented abstract interpretation methods for analyzing such programs using *controlled partitioning*, *i.e.* some Boolean states are encoded explicitly as locations in the CFG, whereas the other Boolean variables are handled symbolically. Pursuing this approach, we have developed methods for applying abstract acceleration to such a *logico-numerical* CFG (see Schrammel and Jeannet (2011) for details).

| | size | nbACCel | Nbac |
|---|---|---|---|
| Gate 1 | 7 | **0.73** | ? |
| Escalator 1 | 12 | **0.49** | ? |
| Traffic 1 | 18 | **0.19** | 3.49 |
| Traffic 2 | 18 | **0.35** | ? |
| LCM Quest 0a-1 | 7 | **0.04** | 0.05 |
| LCM Quest 0a-2 | 6 | **0.05** | 0.19 |
| LCM Quest 0b-1 | 19 | **0.08** | ? |
| LCM Quest 0b-2 | 17 | **0.20** | ? |
| LCM Quest 0c-1 | 28 | **0.16** | 0.86 |
| LCM Quest 0c-2 | 25 | **0.24** | 14.8 |
| LCM Quest 1-1 | 114 | **0.92** | 2.45 |
| LCM Quest 2-1 | 247 | **7.84** | 12.8 |
| LCM Quest 3-1 | 483 | 8.49 | **3.76** |
| LCM Quest 3b-1 | 1724 | 43.8 | **19.1** |
| LCM Quest 3c-1 | 1319 | **34.2** | ? |
| LCM Quest 3d-1 | 281 | **5.43** | ? |
| LCM Quest 3e-1 | 638 | **20.6** | ? |
| LCM Quest 4-1 | 4482 | 186 | **50.1** |

**Table 1.** Experimental comparison between nbACCel and Nbac (size. . . number of locations of the enumerated CFG; times in seconds; ?. . . property not proved)

*Pre-processing.* Some transformations are necessary in order to prepare a self-loop in a CFG for abstract acceleration. These comprise especially the elimination of Boolean input variables resulting in a non-deterministic CFG and splitting non-convex guards into convex ones. Methods for dealing with multiple self-loops in a single location are described by Gonnord (2007).

*Experimental results.* Table 1 shows some experimental results comparing our tool nbACCel, which implements the methods presented in this article, with the tool Nbac (Jeannet (2003)). The benchmarks have quite different sizes from ten up to a few hundred lines of Lustre code. The safety properties we want to prove about the small examples require a very good precision on the numerical variables. We observe that we are able to prove some benchmarks that are not provable using Nbac. This is due to the precision gained by abstract acceleration. But also on the other examples nbACCel is mostly faster than Nbac; except for some of the larger benchmarks where the more sophisticated dynamic partitioning techniques employed in Nbac start to pay off. Further results including a comparison with Aspic (Gonnord (2009)) can be found in Schrammel and Jeannet (2011).

## 8. Related Work

As already outlined in the introduction, there are essentially two approaches to the reachability problem: Those that aim at computing exact results using additive integer (Presburger) or real arithmetic without guarantee of termination (in the cases mentioned in the introduction); and those that are based on abstract interpretation which terminate but generally ignore divisibility properties and deliver only conservative approximations.

*Presburger-based approaches.* Bultan et al. (1997) use Presburger formulas to represent the state space of integer variables in symbolic model checking of concurrent programs, but they use an abstract-interpretation-like widening operator in order to accelerate loops and actually compute an over-approximation.

Boigelot and Wolper (1994); Boigelot (1999) introduce a representation called *periodic vector sets* of the form $\exists \boldsymbol{k} \in \mathbb{Z}^m : \boldsymbol{x} = \mathbf{C}\boldsymbol{k}+\boldsymbol{d} \wedge \mathbf{P}\boldsymbol{k} \leq \boldsymbol{q}$ for capturing the transitive closure of affine transformations, and apply it to communication protocols involving integer variables. Their method tries to accelerate selected cycles in so-called *meta-transitions*, but without guarantee for termination.

Fribourg and Olsén (1997) compute the reachable state space of Petri nets with unbounded markings using a decision procedure over Presburger arithmetic.

Finkel and Leroux (2002); Bardin et al. (2004) apply Presburger-based acceleration to communication protocols. They extend the results of Boigelot (1999) to periodic affine transformations as defined in Sec. 3. Their methods have also been applied to FIFO queues using subclasses of regular expressions (Boigelot and Godefroid (1997); Abdulla et al. (2004)). Bardin et al. (2003) describes the tool FAST for analyzing *flattable* programs, *i.e.* without nested loops. Bardin et al. (2005) present an overall framework for various classes of systems that can be treated by exact acceleration.

Bozga et al. (2010) unifies the computation of transitive closures of such as difference relations, octagonal relations and finite monoid affine transformations by defining the notion of *ultimately periodic relations*. Their method is also based on Presburger arithmetic. Like us, this work investigates the case of relations instead of deterministic functions. Difference relations (and their octogonal generalization) have an incomparable expressive power compared to translations/resets with inputs and simple guards seen as relations:

- Difference relations can express for instance variables permutation, unlike translations/resets;
- but they cannot express translations/resets involving guards with general linear constraints, such as $(x_1+2x_2\leq 10) \rightarrow \left| \begin{array}{l} x'_1=x_1+1 \\ x'_2=x_2+2 \end{array} \right.$ or $(\xi_1+2\xi_2=3) \rightarrow \left| \begin{array}{l} x'_1=x_1+\xi_1 \\ x'_2=x_2+\xi_2 \end{array} \right.$

Last, finite monoid affine transformations, which are deterministic, correspond to the periodic case that we did not consider in Sec. 3.

Acceleration has also been used in the analysis of *timed and hybrid automata*. For example, the concepts of Boigelot (1999) were developed further by Boigelot et al. (2003) to the acceleration of general affine relations $R(\boldsymbol{x}, \boldsymbol{x}')$. The basic technique applies to relations of the form $G(\boldsymbol{x}) \wedge G'(\boldsymbol{x}') \wedge P(\boldsymbol{x}' - \boldsymbol{x})$ where $G$, $G'$ and $P$ are conjunctions of linear inequalities. Observe that, if $G'$ is true, such relations can be expressed in our setting as pure translations with inputs and simple guards, defined by $G(\boldsymbol{x}) \wedge P(\boldsymbol{\xi}) \rightarrow \boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{\xi}$. Boigelot and Herbreteau (2006) develop transformation methods (based mainly on variable changes) to reduce subclasses of affine relations to this basic case. It would be interesting to apply and adapt their methods to abstract acceleration. These two papers apply this approach to the verification of rectangular hybrid systems (as defined by Alur et al. (1995)) that include timed automata.

Recent research exhibited results on the transitive closure of *affine integer tuple relations* (Beletska et al. (2009)). Their goal is to compute multiple clause relations, *i.e.* several self-loops, with commutative transition relations: $\tau_1 \circ \tau_2 = \tau_2 \circ \tau_1$, *e.g.* $\tau_1 : x'_1 = x_1 + 1; x'_2 = 2x_2$ and $\tau_2 : x'_1 = x_1 + 3; x'_2 = 5x_2$. They use Presburger-based methods and apply their results in the context of program parallelization.

*Abstract interpretation-based approaches* to acceleration are comparatively recent. Also our method falls into this category and extends the work of Gonnord and Halbwachs (2006) and Gonnord (2007). Apart from self-loops of translations and translations with resets for which we developed our extensions, they also deal with periodic affine transformations and some special cases of multiple self-loops. Furthermore, they describe methods for unfolding multiple self-loops in order to compute the fixed point more efficiently.

Forward acceleration

| | |
|---|---|
| Translations | $\tau^{\otimes}(X) = X \sqcup \tau\big((X \sqcap G) \nearrow D\big)$ |
| Translations with resets | $\tau^{\otimes}(X) = X \sqcup \tau(X) \sqcup \tau\Big(\big((\tau(X) \sqcap G)^{t,0} \nearrow D^{t,0}\big) + D^{0,r}\Big)$ |

Backward acceleration

| | |
|---|---|
| Translations | $\tau^{-\otimes}(X') = X' \sqcup \big((\tau^{-1}(X') \nearrow (-D)\big) \sqcap G$ |
| Translations with resets | $\tau^{-\otimes}(X') = X' \sqcup \tau^{-1}(X') \sqcup \tau^{-1}\Big(\big((\tau^{-1}(X')^{t,\bullet} \nearrow (-D^{t,0})\big) \sqcap G\Big)$ |

with

$$\tau \;:\; (\overbrace{\mathbf{A}\boldsymbol{x} + \mathbf{L}\boldsymbol{\xi} \leq \boldsymbol{b}}^{G_0(\boldsymbol{x},\boldsymbol{\xi})} \wedge \overbrace{\mathbf{J}\boldsymbol{\xi} \leq \boldsymbol{k}}^{G_1(\boldsymbol{\xi})}) \longrightarrow \boldsymbol{x}' = \mathbf{C}\boldsymbol{x} + \mathbf{T}\boldsymbol{\xi} + \boldsymbol{u} \quad \mathbf{C} \text{ diagonal matrix with 0 or 1 only}$$

$$G(\boldsymbol{x}) = \exists \boldsymbol{\xi} : G_0(\boldsymbol{x}, \boldsymbol{\xi}) \qquad D = \{\boldsymbol{d} \mid \exists \boldsymbol{\xi} : \boldsymbol{d} = \boldsymbol{T}\boldsymbol{\xi} + \boldsymbol{u} \wedge G_1(\boldsymbol{\xi}) \wedge \exists \boldsymbol{x} : G_0(\boldsymbol{x}, \boldsymbol{\xi})\}$$

Approximations

| | |
|---|---|
| In all cases | dense and convex approximation |
| $L \neq \mathbf{0}$ | $G$ and $D$ are decoupled |
| Translations with resets | $D$ approximated by the Cartesian product $D^t \times D^r$ |

**Table 2.** Overview of abstract acceleration formulas

The affine derivative closure algorithm (Ancourt et al. (2010)) that we explained in more detail in Section 6.3 is also based on abstract interpretation methods, but unlike abstract acceleration it neither inspects the types of transitions in order to apply an optimized acceleration formula nor specializes w.r.t. a given initial state.

## 9. Conclusion and Discussion

We have presented an extension of abstract acceleration to numerical inputs for forward and backward analysis. Table 2 shows a summary of the formulas. This extension is less straightforward than supposed – most notably due to the observation that inputs can be used to turn translations into arbitrary affine transformations; also, resetting variables to input values may cause some subtle behavior. Regarding approximations, Table 2 shows the cases where our method is precise in the sense that we perform only dense and convex approximations, and the more complex cases for which additional approximations are necessary to abstract away the number of iterations.

Abstract acceleration elegantly integrates into an abstract interpretation-based verification tool, where it is usually used in combination with widening: as pointed out in Sec. 6.2, due to its monotonicity property it is possible to accelerate the innermost loops precisely while using widening for the outer loops in nested loop situations. Thus, much better invariants can be computed for programs where a lot of information is lost when using widening only.

In comparison to other abstract interpretation-based transitive closure methods, for instance the affine derivative closure algorithm of Ancourt et al. (2010), abstract acceleration deals only with some frequently occurring types of self-loop transitions, for which

it yields more precise results (especially for transitions with resets), but it needs to resort to widening in the general case.

We have reported some first experimental results that give evidence about the potential of abstract acceleration w.r.t. improving reachability analysis in terms of precision and performance.

Regarding future work, we could extend slightly the transformations we consider. Firstly, we conjecture that it is possible to generalize our results from the case where $C$ is a diagonal matrix with either 0 or 1 to the case where $C$ is periodic with prefix $p$ and period $l$ as defined in Sec. 3. This could be done in the same way as Finkel and Leroux (2002) by rewriting $\bigcup_{k\geq 0} \tau^k$ as

$$\tau^0 \cup \ldots \cup \tau^{p-1} \cup \bigcup_{k\geq 0} (\tau^l)^k \circ \tau^p \cup \ldots \cup \bigcup_{k\geq 0} (\tau^l)^k \circ \tau^{p+l-1}$$

and accelerating $\tau^l$ on the image of $\tau^{p+i}$ with $0 \leq i \leq l-1$.

Secondly, it should be also possible to generalize and unify the two classes of relations induced resp. by forward and backward translations/resets with inputs and simple guards to a more general symmetrical class, and to consider the acceleration of such relations. A candidate could be

$$R(\boldsymbol{x},\boldsymbol{x}') \Leftrightarrow \left\{ \begin{array}{c} \mathbf{A}\boldsymbol{x} \leq \boldsymbol{b} \wedge \mathbf{J}\boldsymbol{\xi} \leq k \\ \mathbf{A}'\boldsymbol{x}' \leq \boldsymbol{b}' \end{array} \right. \wedge \left\{ \begin{array}{rcll} \boldsymbol{x}'_t - \boldsymbol{x}_t & = & \mathbf{T}_t\boldsymbol{\xi} + \boldsymbol{u}_t & \text{(translated dimensions)} \\ \boldsymbol{x}'_f & = & \mathbf{T}_f\boldsymbol{\xi} + \boldsymbol{u}_f & \text{(forward reset dimensions)} \\ \boldsymbol{x}_b & = & \mathbf{T}_b\boldsymbol{\xi} + \boldsymbol{u}_b & \text{(backward reset dimensions)} \end{array} \right.$$

in which the set of dimensions is partioned into three classes.

Regarding integer (*e.g.,* divisibility) properties, our techniques based on convex polyhedra cannot express them and Remark 2 discusses the effect of the induced dense approximation. To improve on this we could combine our techniques with the linear congruence abstract domain introduced in Granger (1991). This domain satisfies the finite ascending condition, hence it does not require widening nor acceleration. By this means we are able to tighten the results. For instance, if the abstract acceleration results in a convex polyhedron $2 \leq x_1 \leq 8 \wedge 2x_1 + x_2 \leq 9$, and we know from the linear congruences domain that $x_1 = 1 \bmod 4 \wedge x_2 = 0 \bmod 2$ (typically because $(x_1, x_2)$ are iteratively translated by $(4, 2)$ from a known value), then the convex polyhedron can be tightened to $x_1 = 5 \wedge 2x_1 + x_2 \leq 8$. Compared to Presburger arithmetic, we still limit ourselves to convex sets with such a technique. The general goal would be to add congruence constraints in the guard of the transformation $\tau$ and to exploit them as sketched above.

The question under which condition translations/resets with inputs viewed as relations are ultimately periodic relations in the sense of Bozga et al. (2010) could be also investigated. We conjecture that this is only true in the cases where our methods perform only dense and convex approximations (see Table 2).

## References

Abdulla, P. A., Collomb-Annichini, A., Bouajjani, A., Jonsson, B., 2004. Using forward reachability analysis for verification of lossy channel systems. Formal Methods in System Design 25 (1), 39–65.

Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T., Ho, P., Nicollin, X., Olivero, A., Sifakis, J., Yovine, S., 1995. The algorithmic analysis of hybrid systems. Theoretical Computer Science 138 (1), 3–34.

Ancourt, C., Coelho, F., Irigoin, F., 2010. A modular static analysis approach to affine loop invariants detection. In: Numerical and Symbolic Abstract Domains. Vol. 267 of ENTCS. Elsevier, pp. 3–16.

Bagnara, R., Dobson, K., Hill, P. M., Mundell, M., Zaffanella, E., 2006. Grids: A domain for analyzing the distribution of numerical values. In: Logic-Based Program Synthesis and Transformation, LOPSTR'06. Vol. 4407 of LNCS. pp. 219–235.

Bagnara, R., Hill, P. M., Ricci, E., Zaffanella, E., 2003. Precise widening operators for convex polyhedra. In: Static Analysis Symposium, SAS'03. LNCS. Springer, pp. 337–354.

Bardin, S., Finkel, A., Leroux, J., 2004. Faster acceleration of counter automata in practice. In: Tools and Algorithms for the Construction and Analysis of Systems. Vol. 2988 of LNCS. pp. 576–590.

Bardin, S., Finkel, A., Leroux, J., Petrucci, L., 2003. FAST: Fast acceleration of symbolic transition systems. In: Computer Aided Verification. Vol. 2725 of LNCS. Springer, pp. 118–121.

Bardin, S., Finkel, A., Leroux, J., Schnoebelen, P., 2005. Flat acceleration in symbolic model checking. In: Automated Technology for Verification and Analysis. Vol. 3707 of LNCS. Springer, pp. 474–488.

Beletska, A., Barthou, D., Bielecki, W., Cohen, A., 2009. Computing the transitive closure of a union of affine integer tuple relations. In: Combinatorial Optimization and Applications. Vol. 6508 of LNCS. Springer, pp. 98–109.

Benoy, F., King, A., Mesnard, F., 2005. Computing convex hulls with a linear solver. Theory and Practice of Logic Programming 5 (1-2), 259–271.

Boigelot, B., 1999. Symbolic methods for exploring infinite state spaces. Phd thesis, University of Liège.

Boigelot, B., Godefroid, P., 1997. Symbolic verification of communication protocols with infinite state spaces using QDDs. Formal Methods in System Design 14 (3), 237–255.

Boigelot, B., Herbreteau, F., 2006. The power of hybrid acceleration. In: Computer Aided Verification. Vol. 4144 of LNCS. Springer, pp. 438–451.

Boigelot, B., Herbreteau, F., Jodogne, S., 2003. Hybrid acceleration using real vector automata (extended abstract). In: Computer Aided Verification. Vol. 2725 of LNCS. pp. 193–205.

Boigelot, B., Wolper, P., 1994. Symbolic verification with periodic sets. In: Computer Aided Verification. Vol. 818 of LNCS. Springer, pp. 55–67.

Bozga, M., Iosif, R., Konecný, F., 2010. Fast acceleration of ultimately periodic relations. In: Computer Aided Verification. Vol. 6174 of LNCS. Springer, pp. 227–242.

Bultan, T., Gerber, R., Pugh, W., 1997. Symbolic model checking of infinite state systems using presburger arithmetic. In: Computer Aided Verification. Vol. 1254 of LNCS. Springer, pp. 400–411.

Caspi, P., Pilaud, D., Halbwachs, N., Plaice, J. A., 1987. LUSTRE: a declarative language for real-time programming. In: Principles of Programming Languages. ACM, pp. 178–188.

Comon, H., Jurski, Y., 1998. Multiple counters automata, safety analysis and presburger arithmetic. In: Computer Aided Verification. Vol. 1427 of LNCS. Springer, pp. 268–279.

Cousot, P., Cousot, R., 1977. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Principles of Programming Languages. pp. 238–252.

Cousot, P., Cousot, R., 1992a. Abstract interpretation and application to logic programs. Journal of Logic Programming 13 (2–3), 103–179.

Cousot, P., Cousot, R., 1992b. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation. In: Programming Language Implementation and Logic Programming. Vol. 631 of LNCS. Springer, pp. 269–295.

Cousot, P., Halbwachs, N., 1978. Automatic discovery of linear restraints among variables of a program. In: Principles of Programming Languages. ACM, pp. 84–97.

de Berg, M., Cheong, O., van Kreveld, M., Overmars, M., 2008. Computational Geometry: Algorithms and Applications, 3rd Edition. Springer.

Finkel, A., Leroux, J., 2002. How to compose Presburger-accelerations: Applications to broadcast protocols. In: Foundations of Software Technology and Theoretical Computer Science. Vol. 2556 of LNCS. Springer, pp. 145–156.

Fribourg, L., Olsén, H., 1997. Proving safety properties of infinite state systems by compilation into presburger arithmetic. In: Conference on Concurrency Theory. Vol. 1243 of LNCS. Springer, pp. 213–227.

Fukuda, K., Prodon, A., 1996. Double description method revisited. In: Combinatorics and Computer Science. Vol. 1120 of LNCS. Springer, pp. 91–111.

Gonnord, L., 2007. Accélération abstraite pour l'amélioration de la précision en analyse des relations linéaires. Phd thesis, Université Joseph Fourier, Grenoble.

Gonnord, L., 2009. The ASPIC tool: Accelerated symbolic polyhedral invariant computation, `http://laure.gonnord.org/pro/aspic/aspic.html`.

Gonnord, L., Halbwachs, N., 2006. Combining widening and acceleration in linear relation analysis. In: Static Analysis Symposium. Vol. 4134 of LNCS. Springer, pp. 144–160.

Granger, P., 1991. Static analysis of linear congruence equalities among variables of a program. In: TAPSOFT'91. Vol. 493 of LNCS. pp. 169–192.

Halbwachs, N., Lagnier, F., Raymond, P., 1993. Synchronous observers and the verification of reactive systems. In: Algebraic Methodology and Software Technology. Workshops in Computing. Springer, pp. 83–96.

Halbwachs, N., Proy, Y.-E., Roumanoff, P., 1997. Verification of real-time systems using linear relation analysis. Formal Methods in System Design 11 (2), 157–185.

Jeannet, B., 2003. Dynamic partitioning in linear relation analysis. application to the verification of reactive systems. Formal Methods in System Design 23 (1), 5–37.

Jeannet, B., Jéron, T., Rusu, V., Zinovieva, E., 2005. Symbolic test selection based on approximate analysis. In: Tools and Algorithms for the Construction and Analysis of Systems. Vol. 3440 of LNCS. Springer, pp. 349–364.

Jones, N. D., Gomard, C., Sestoft, P., 1993. Partial Evaluation and Automatic Program Generation. Prentice Hall International.

Kam, J. B., Ullman, J. D., 1977. Monotone data flow analysis frameworks. Acta Informatica 7, 305–317.

Leroux, J., 2003. Algorithmique de la vérification des systèmes à compteurs – approximation et accélération – implémentation dans l'outil FAST. Phd thesis, École Normale Supérieure de Cachan.

Minsky, M. L., 1961. Recursive unsolvability of post's problem of 'tag' and other topics in theory of turing machines. Annals of Mathematics 74 (3), 437–455.

Schrammel, P., Jeannet, B., 2010. Extending abstract acceleration to data-flow programs with numerical inputs. In: Numerical and Symbolic Abstract Domains. Vol. 267 of ENTCS. Elsevier, pp. 101–114.

Schrammel, P., Jeannet, B., 2011. Logico-numerical abstract acceleration and application to the verification of data-flow programs. In: Static Analysis Symposium, SAS'11. Vol. 6887 of LNCS. Springer, pp. 233–248.